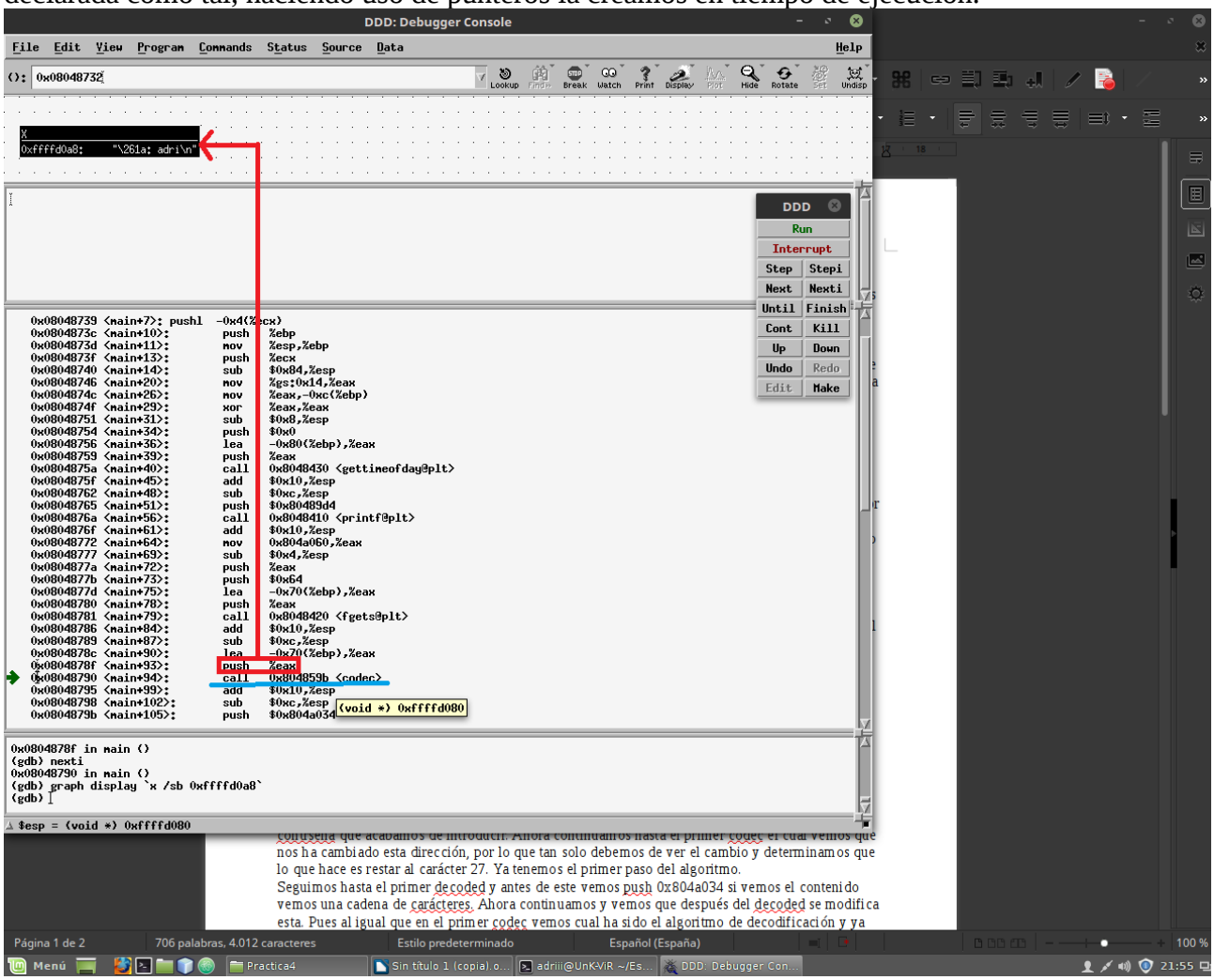
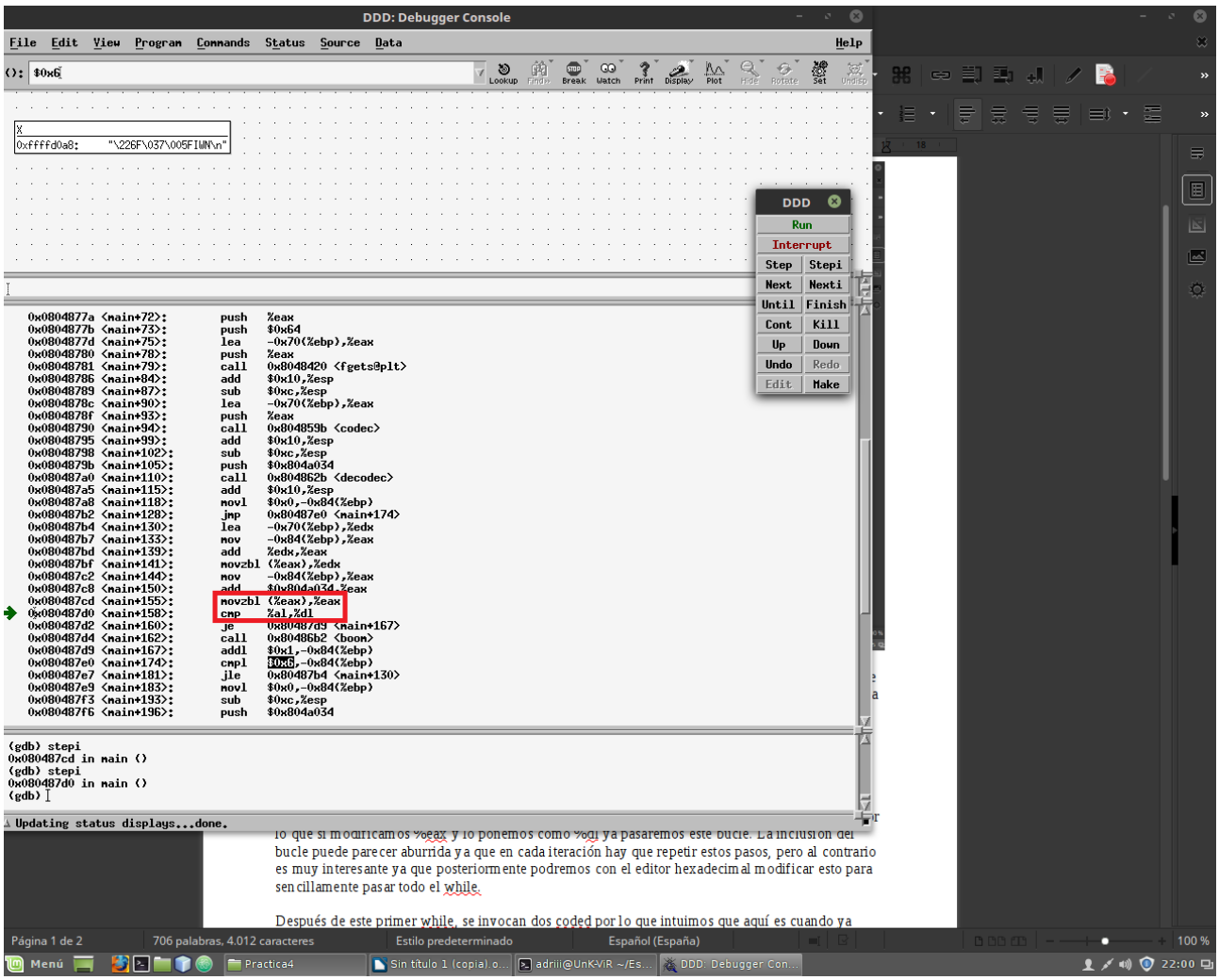
**Desactivación de mi bomba:**

Para desactiva mi bomba, comenzamos poniendo un breakpoint al principio del programa, después de esto vamos ejecutando con stepi y nos daremos cuenta de que se llama a la función codec.
Lo que hace esta función es generar la contraseña la cual siempre es la misma pero no esta declarada como tal, haciendo uso de punteros la creamos en tiempo de ejecución.




Una vez vamos avanzando vemos que nos encontramos con un bucle el cual parece ser que recorre toda la palabra comprobando la introducida con la contraseña, sabemos que dentro de la cabecera a cada iteración hay una variable que aumenta 1 hasta el tamaño de la palabra, a todo esto hemos visto que se ha invocado a la función codec y decoded por lo que intuimos que estas funciones codifican y decodifican las contraseñas y que al parecer como hemos visto al principio la codec se llama varias veces por lo que dentro tendrá ifs y variables estáticas para impedir la ejecución de código repetido.
Para saltarse el primer bucle tan solo hay ir paso por paso y y cuando lleguemos a cmp vemos que en la línea anterior tenemos el valor de %al que esta contenido en %eax, son registros de 8 bits, por lo que si modificamos %eax y lo ponemos como %dl ya pasaremos este bucle. La inclusión del bucle puede parecer aburrida ya que en cada iteración hay que repetir estos pasos, pero al contrario es muy interesante ya que posteriormente podremos con el editor hexadecimal modificar esto para sencillamente pasar todo el while.

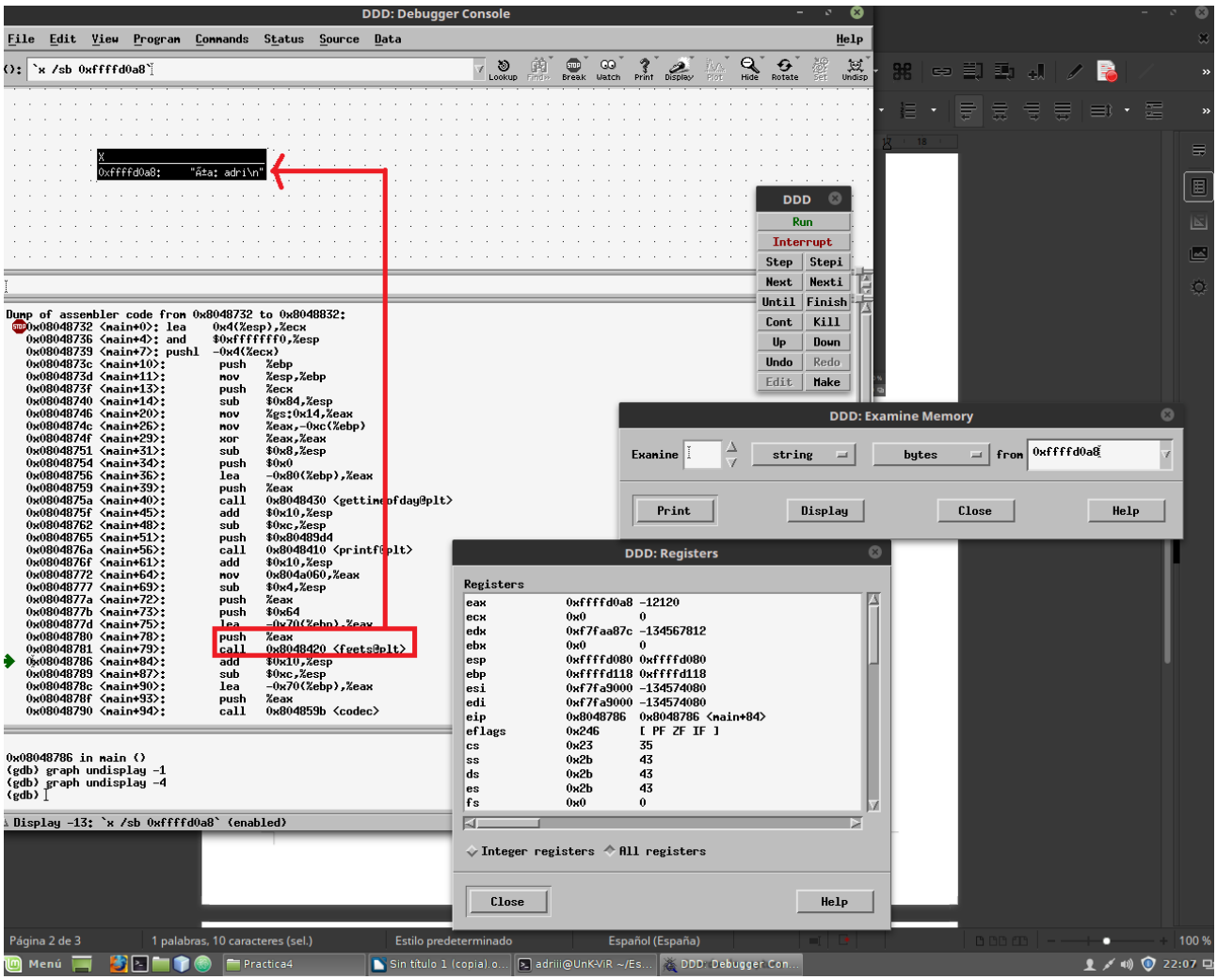



Después de este primer while, se invocan dos coded por lo que intuimos que aquí es cuando ya ambas contraseñas están iguales y es cuando de nuevo se entra en un bucle como el anterior el cual se salta con la misma facilidad que el anterior.

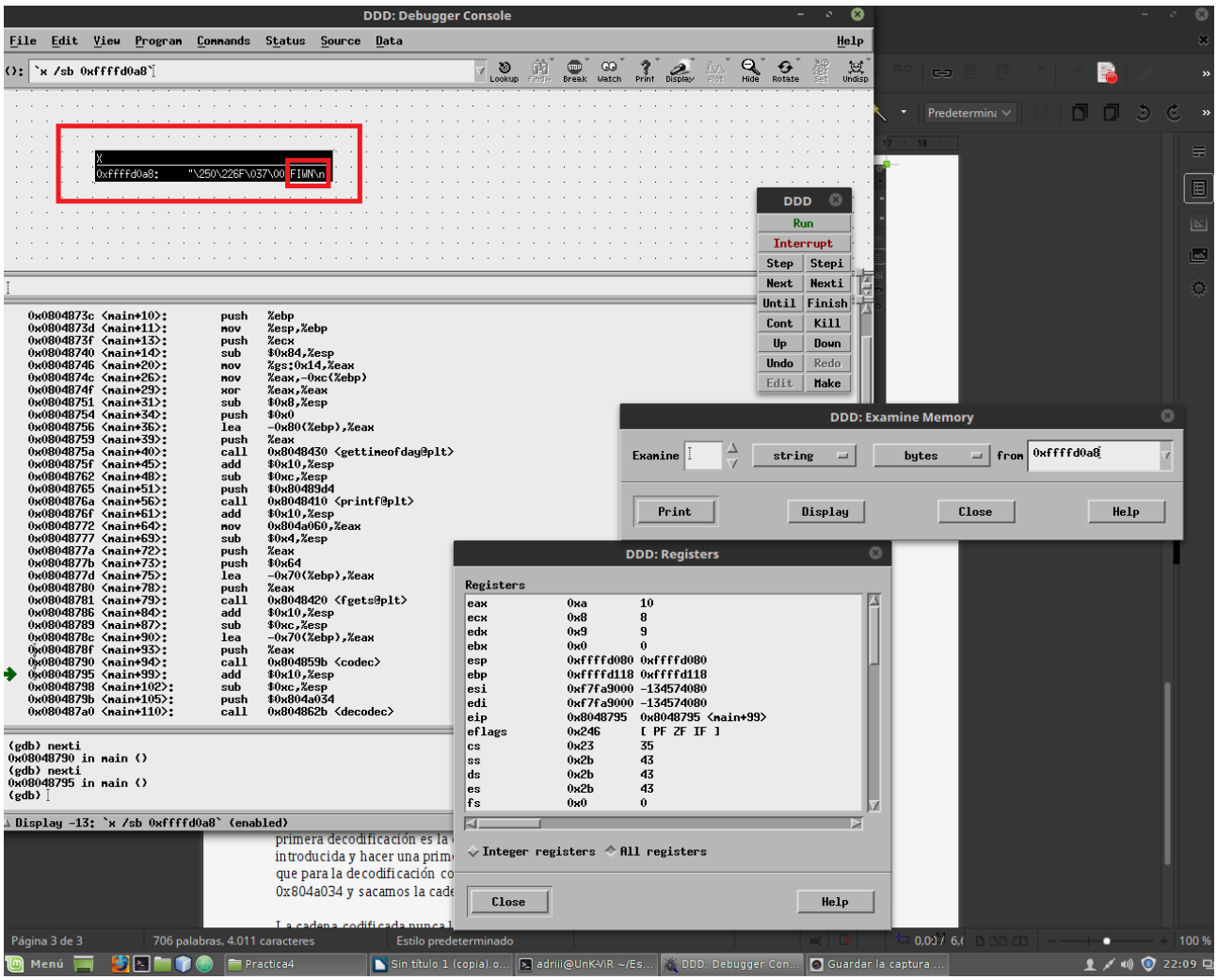
Una vez pasado esto ya solo queda la verificación de código numérico, el cual si nos fijamos en el ddd podemos ver como hay una adición de 1 por lo que intuimos que la contraseña original será 1 más que la correcta que pongamos. Saltarlo es tan fácil como igualar los valores en el cmp.

Una vez hecho lo anterior habremos desactivado la bomba sin saber ni la contraseña ni el código numérico.

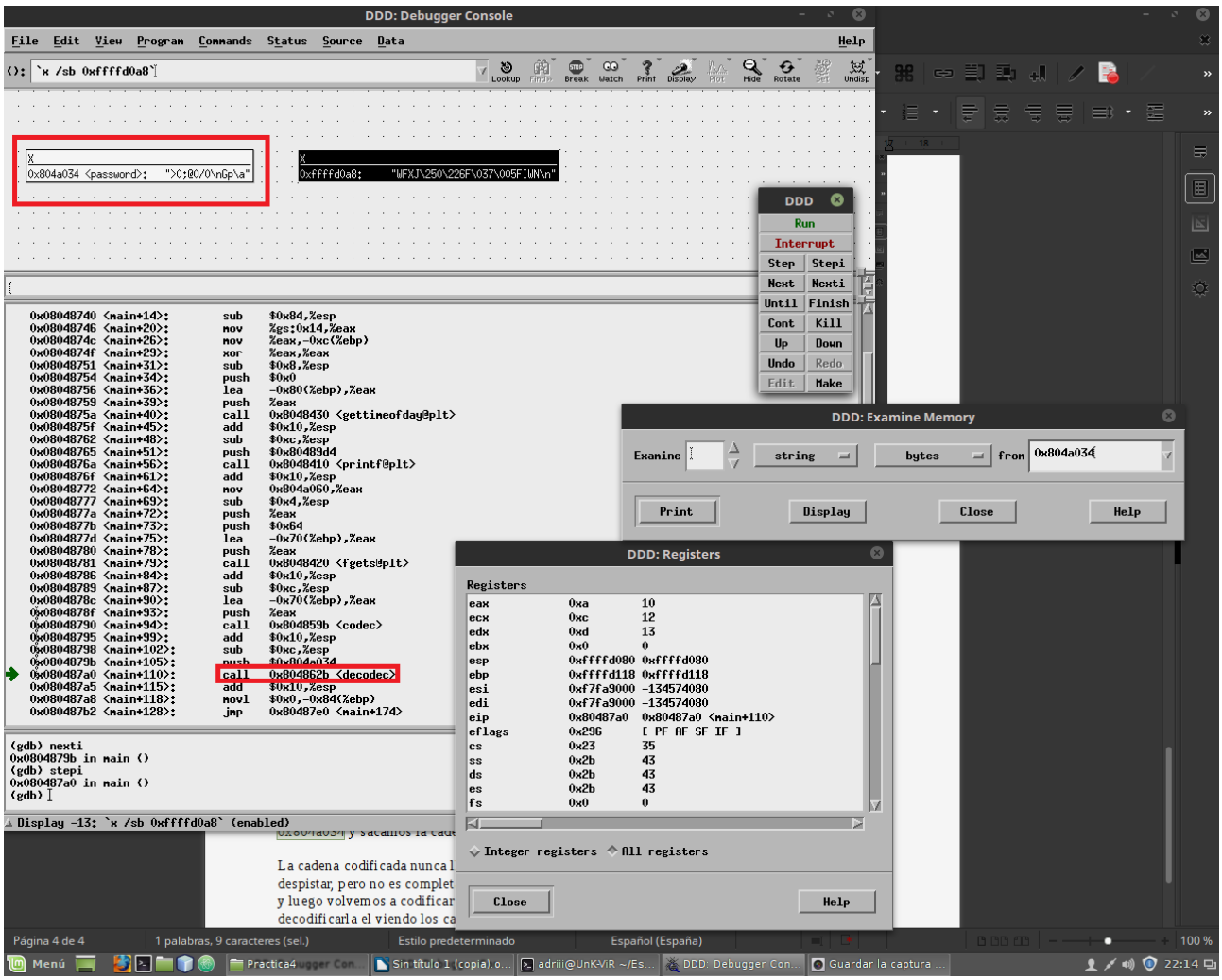
Ahora bien si queremos sacar la contraseña debemos de investigar direcciones de memoria, para empezar investigamos la dirección de eax después del fgets 0xffffd0a8, esta es la dirección de la contraseña que acabamos de introducir.

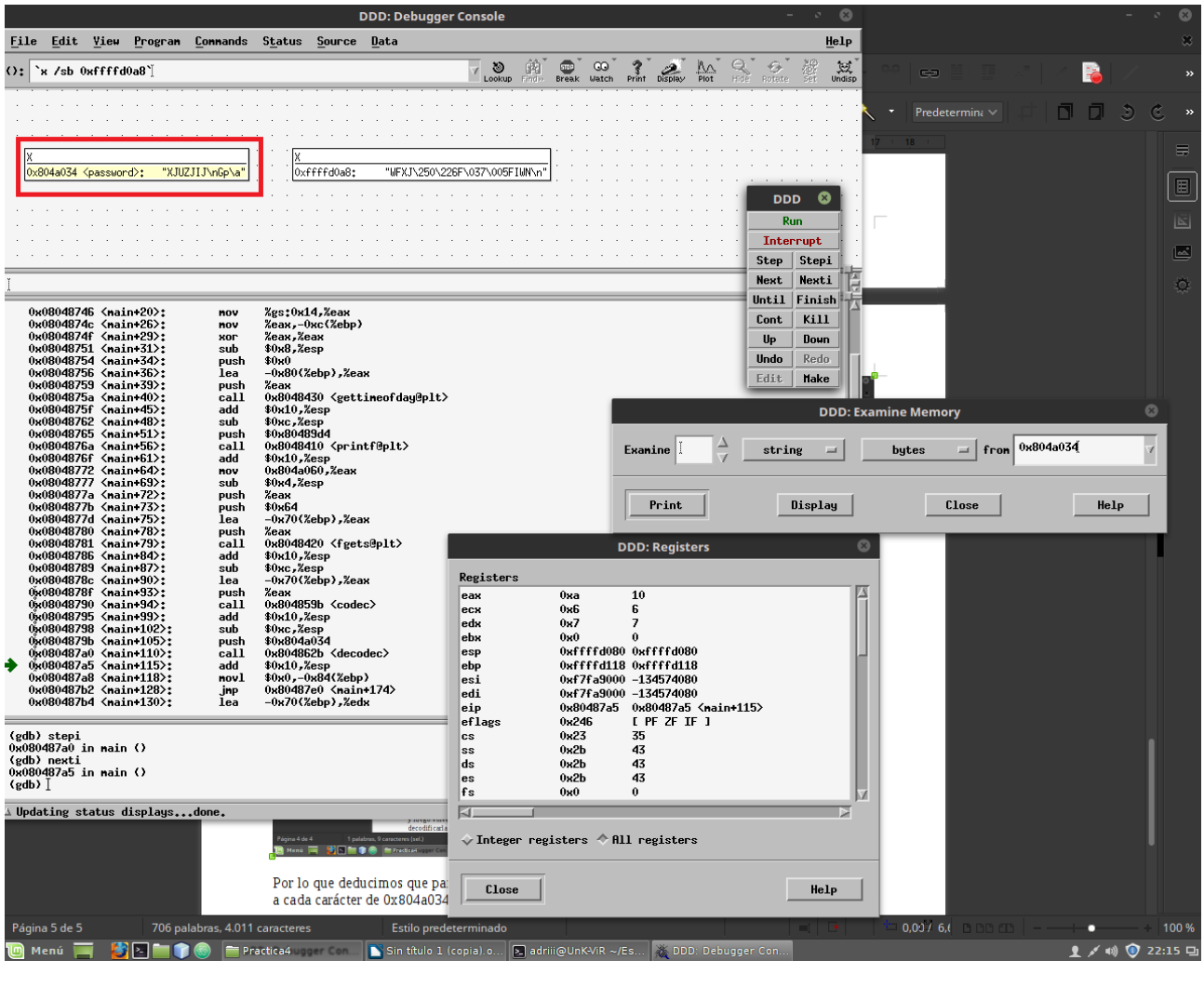



Ahora continuamos hasta el primer codec el cual vemos que nos ha cambiado esta dirección, por lo que tan solo debemos de ver el cambio y determinamos que lo que hace es restar al carácter 27. Ya tenemos el primer paso del algoritmo.




Seguimos hasta el primer decoded y antes de este vemos push 0x804a034 si vemos el contenido vemos una cadena de caracteres. Ahora continuamos y vemos que después del decoded se modifica esta. Pues al igual que en el primer codec vemos cual ha sido el algoritmo de decodificación y ya tenemos todo. Sabemos que codifica restando 27 que lo hemos visto dentro del primer codec al ver como quedaba nuestra cadena introducida y que en la decodificación suma 26 por lo que si lo pensamos nos damos cuenta de que si para codificar resta pues para codificar sumamos y que la primera decodificación es la opuesta a la primera codificación de esta manera al codificar la cadena introducida y hacer una primera decodificación tenemos la misma cadena.


DDD: Debugger Console
File Edit View Program Commands Status Source Data Help
(): `x /sb 0xffffd0a8`
0x804a034 <password>: ">0;@0/0\nGp\a"
0xffffd0a8: "WFXJ\250\226F\037\005FIWN\n"
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
0x08048740 <main+14>: sub $0x84,%esp
0x08048746 <main+20>: mov %gs:0x14,%eax
0x0804874c <main+26>: mov %eax,-0xc(%ebp)
0x0804874f <main+29>: xor %eax,%eax
0x08048751 <main+31>: sub $0x8,%esp
0x08048754 <main+34>: push $0x0
0x08048756 <main+36>: lea -0x80(%ebp),%eax
0x08048759 <main+39>: push %eax
0x0804875a <main+40>: call 0x8048430 <gettimeofday@plt>
0x0804875f <main+45>: add $0x10,%esp
0x08048762 <main+48>: sub $0xc,%esp
0x08048765 <main+51>: push $0x80489d4
0x0804876a <main+56>: call 0x8048410 <printf@plt>
0x0804876f <main+61>: add $0x10,%esp
0x08048772 <main+64>: mov 0x804a060,%eax
0x08048777 <main+69>: sub $0x4,%esp
0x0804877a <main+72>: push %eax
0x0804877b <main+73>: push $0x64
0x0804877d <main+75>: lea -0x70(%ebp),%eax
0x08048780 <main+78>: push %eax
0x08048781 <main+79>: call 0x8048420 <fgets@plt>
0x08048786 <main+84>: add $0x10,%esp
0x08048789 <main+87>: sub $0xc,%esp
0x0804878c <main+90>: lea -0x70(%ebp),%eax
0x0804878f <main+93>: push %eax
0x08048790 <main+94>: call 0x804859b <codec>
0x08048795 <main+99>: add $0x10,%esp
0x08048798 <main+102>: sub $0xc,%esp
0x0804879b <main+105>: push $0x804a034
0x080487a0 <main+110>: call 0x804862b <decodec>
0x080487a5 <main+115>: add $0x10,%esp
0x080487a8 <main+118>: movl $0x0,-0x84(%ebp)
0x080487b2 <main+128>: jmp 0x80487e0 <main+174>
(gdb) nexti
0x0804879b in main ()
(gdb) stepi
0x080487a0 in main ()
(gdb)
Display -13: `x /sb 0xffffd0a8` (enabled)
DDD: Examine Memory
Examine string bytes from 0x804a034
Print Display Close Help
DDD: Registers
Registers
eax 0xa 10
ecx 0xc 12
edx 0xd 13
ebx 0x0 0
esp 0xffffd080 0xffffd080
ebp 0xffffd118 0xffffd118
esi 0xf7fa9000 -134574080
edi 0xf7fa9000 -134574080
eip 0x80487a0 0x80487a0 <main+110>
eflags 0x296 [ PF AF SF IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
Integer registers All registers
Close Help
La cadena codificada nunca l
despistar, pero no es complet
y luego volvemos a codificar
decodificarla el viendo los ca
Página 4 de 4 1 palabras, 9 caracteres (sel.) Estilo predeterminado Español (España) 100 %
Menú Practica4ugger Con... Sin título 1 (copia) o... adrii@UnKViR ~/Es... DDD: Debugger Con... Guardar la captura ... 22:14


DDD: Debugger Console
File Edit View Program Commands Status Source Data Help
(): `x /sb 0xffffd0a8`
0x804a034 <password>: "XJUZJIJ\nGp\a"
0xffffd0a8: "WFXJ\250\226F\037\005FIWN\n"
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
0x08048746 <main+20>: mov %gs:0x14,%eax
0x0804874c <main+26>: mov %eax,-0xc(%ebp)
0x0804874f <main+29>: xor %eax,%eax
0x08048751 <main+31>: sub $0x8,%esp
0x08048754 <main+34>: push $0x0
0x08048756 <main+36>: lea -0x80(%ebp),%eax
0x08048759 <main+39>: push %eax
0x0804875a <main+40>: call 0x8048430 <gettimeofday@plt>
0x0804875f <main+45>: add $0x10,%esp
0x08048762 <main+48>: sub $0xc,%esp
0x08048765 <main+51>: push $0x80489d4
0x0804876a <main+56>: call 0x8048410 <printf@plt>
0x0804876f <main+61>: add $0x10,%esp
0x08048772 <main+64>: mov 0x804a060,%eax
0x08048777 <main+69>: sub $0x4,%esp
0x0804877a <main+72>: push %eax
0x0804877b <main+73>: push $0x64
0x0804877d <main+75>: lea -0x70(%ebp),%eax
0x08048780 <main+78>: push %eax
0x08048781 <main+79>: call 0x8048420 <fgets@plt>
0x08048786 <main+84>: add $0x10,%esp
0x08048789 <main+87>: sub $0xc,%esp
0x0804878c <main+90>: lea -0x70(%ebp),%eax
0x0804878f <main+93>: push %eax
0x08048790 <main+94>: call 0x804859b <codec>
0x08048795 <main+99>: add $0x10,%esp
0x08048798 <main+102>: sub $0xc,%esp
0x0804879b <main+105>: push $0x804a034
0x080487a0 <main+110>: call 0x804862b <decodec>
0x080487a5 <main+115>: add $0x10,%esp
0x080487a8 <main+118>: movl $0x0,-0x84(%ebp)
0x080487b2 <main+128>: jmp 0x80487e0 <main+174>
0x080487b4 <main+130>: lea -0x70(%ebp),%edx
(gdb) stepi
0x080487a0 in main ()
(gdb) nexti
0x080487a5 in main ()
(gdb)
Updating status displays...done.
DDD: Examine Memory
Examine string bytes from 0x804a034
Print Display Close Help
DDD: Registers
Registers
eax 0xa 10
ecx 0x6 6
edx 0x7 7
ebx 0x0 0
esp 0xffffd080 0xffffd080
ebp 0xffffd118 0xffffd118
esi 0xf7fa9000 -134574080
edi 0xf7fa9000 -134574080
eip 0x80487a5 0x80487a5 <main+115>
eflags 0x246 [ PF ZF IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
Integer registers All registers
Close Help
Por lo que deducimos que pa
a cada carácter de 0x804a034
Página 5 de 5 706 palabras, 4.011 caracteres Estilo predeterminado 100 %
Menú Practica4ugger Con... Sin título 1 (copia) o... adrii@UnKViR ~/Es... DDD: Debugger Con... 22:15

Por lo que deducimos que para la decodificación completa tan solo debemos de sumar 27 + 26 = 53 a cada carácter de 0x804a034 y sacamos la cadena sepuede.

La cadena codificada nunca llega a decodificarse tan solo una vez como paso intermedio para despistar, pero no es completa la decodificación ya que esta tiene dos pasos. Realizamos la primera y luego volvemos a codificar por lo que el alumno nunca vera la clave decodificada, tendrá que decodificarla el viendo los cambios que se realizan.

En el código se ve claramente los dos pasos de codificación y decodificación y se ve que nunca se decodifica completa, tan solo un paso intermedio para despistar y si lo juntas con el primer paso de codificación y a este lo inviertes tienes 27 + 26.

También otra manera es hacer que salte las comparaciones y que se codifique dos veces tu propia cadena y ya viendo los cambios pues tienes que realizar lo inverso en la dirección 0x804a034. Quizás esta última forma es menos engorrosa y sale igualmente tan solo se debe invertir lo que te hagan los decoded a tu clave introducida y aplicárselo a la clave codificada que vemos en las capturas en la dirección  0x804a034