



Software	Crackme Ret0 by Bym24v
Protección	Password.
HERRAMIENTAS	<p>Windows 7 Home Premium SP1 x32 Bits (SO donde trabajamos) OllyDBG v1.10 Exeinfo PE v0.0.5.6 sign 2019.05.22</p> <p><a href="#">DESCARGAR HERRAMIENTAS</a></p> <p><a href="#">DESCARGAR TUTO+ARCHIVOS</a></p>
SOLUCIÓN	PASSWORD. CTF - Capture The Flag
AUTOR	LUISFECAB
RELEASE	Junio 12 2019 [TUTORIAL 016]

# INTRODUCCIÓN

Empezamos con otro nuevo tutorial, el cual es un compromiso adquirido con todos mis amigos de **CracksLatinoS** y **PeruCrackers**, y en especial con todos aquellos que leen mis tutos.

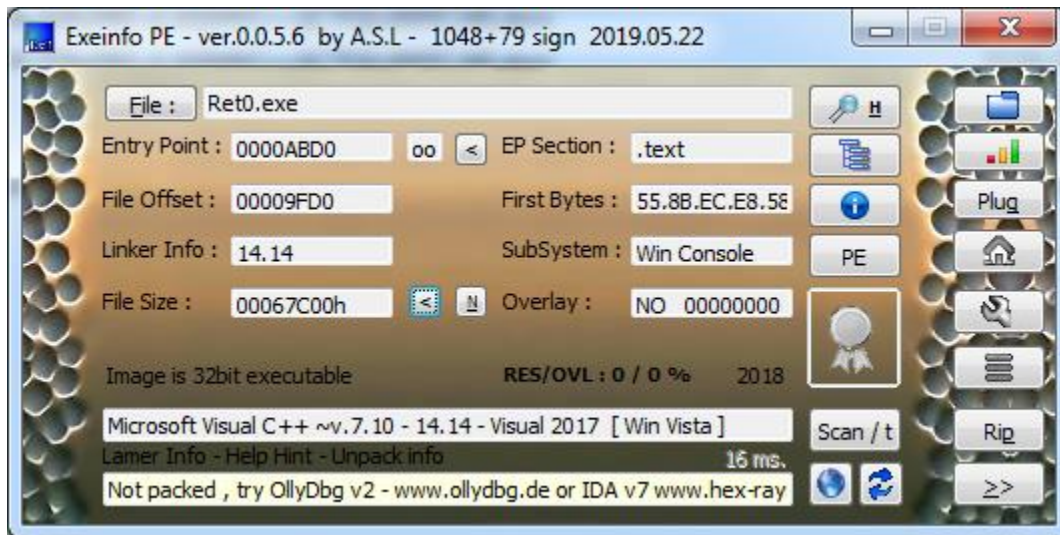
El <Ret0> es nuevo para mí, en el sentido en que lo podemos clasificar como **CTF - Capture The Flag** -. Bueno, es igual a los pocos **Crackmes** que he hecho, solo que al <CHICO BUENO> lo conocemos como la **FLAG** que debemos hallar.

Un reto muy bueno que programó **Bym24v**, que me confundió al inicio pero que poco a poco y de tracear sin parar pude entender por dónde iban las aguas. Siempre he dicho que no soy muy rápido crackeando, pero ahí voy, a mi ritmo y con mucha paciencia contrarresto mi falta de nivel para vencer estos en menos tiempo; que al final no importa porque aquí no hay límite de tiempo y nadie te va a decir algún reclamo si no logras un reto, es todo lo contrario, estarán ahí para ayudarte y darte una voz de aliento para cuando estés por rendirte.

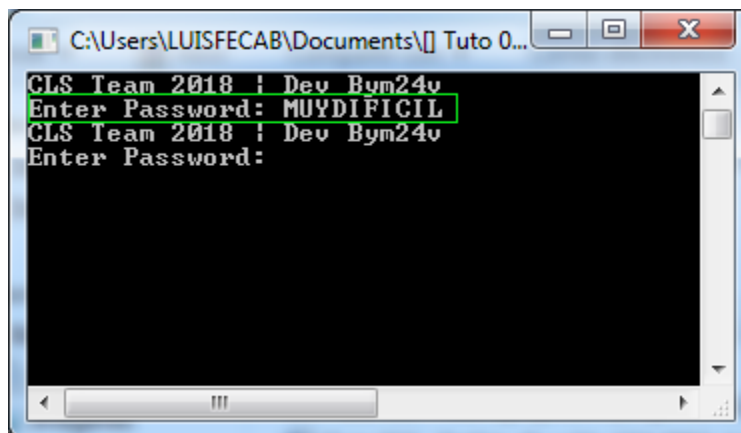
Siempre escribo mis tutos con la esperanza que sean de agrado y de mucha ayuda para quien lo lee, y espero que este no sea la excepción. Espero disfruten de esta lectura.

## ANÁLISIS INICAL

Carguemos el <Ret0> en el <Exeinfo PE v0.0.5.6 sign 2019.05.22> que lo compartió **Apuromafo** hace un par de días atrás.



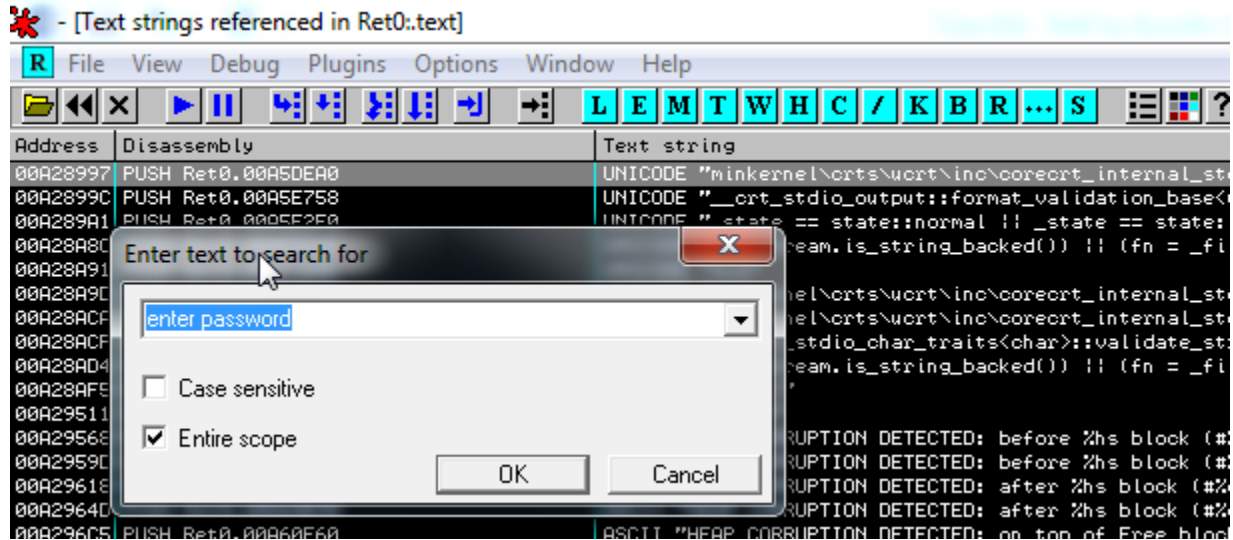
Listo, no está empacado ni nada, es un **MVC++**. Como anécdota cuando **Bym24v** compartió el reto en el canal de **CracksLatinos** dijeron los que saben, que tenía mucho Overhead; por mi parte ni idea a qué se referían.



Al ejecutarlo le metemos mi serial de la buena suerte, "**MUYDIFICIL**" y lo probamos; no muestra nada, solo vuelve al inicio. No hay <**CHICO MALO**> que nos sirva para lograr encontrar la **ZONA CALIENTE** por ahí, pero eso no importa vamos a entrarle de una para ver por dónde le atacamos.

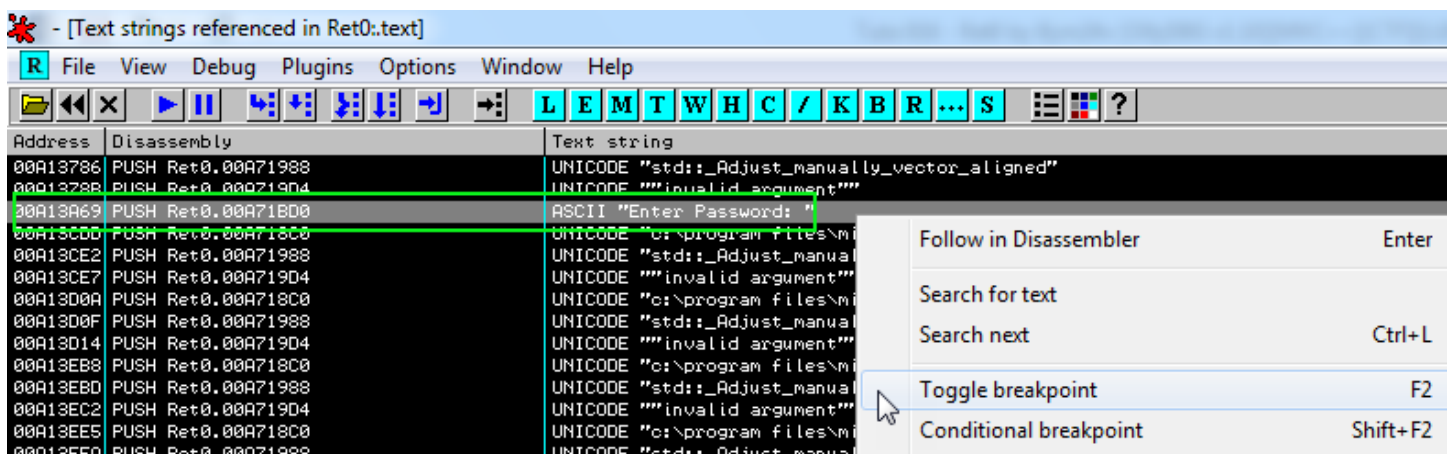
# AL ATAQUE

Carguemos el <Ret0> en el **OllyDBG v1.10** y busquemos las **Strings** por <**All referenced text Strings**>.



Son un montón de **Strings** que parecieran fueran utilizadas para técnicas antidebuging o algo parecido, pero de seguro nada bueno ha de ser, es más, esas **Strings** pueden hacer parte del tal Overhead que comente hace un rato al final de la sección anterior.

Buscaremos la **String** "enter password" porque esa será carga de nuevo cada vez que probemos un **PASSWORD** chueco.



Encontramos la **String** que nos interesa y le ponemos su <**BREAKPOINT**>. Ejecutemos con <**F9**>.

## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISEFCAB]

00A13A65	. 6A 0	PUSH 0	
00A13A67	. 6A 1	PUSH 10	
00A13A69	. 68 D	PUSH Ret0.00A71BD0	ASCII "Enter Password: "
00A13A6E	. 8B88	MOV ECX,DWORD PTR DS:[EAX+A75F70]	
00A13A74	. 8B01	MOV EAX,DWORD PTR DS:[ECX]	
00A13A76	. 8B40	MOV EAX,DWORD PTR DS:[EAX+24]	
00A13A79	. FF00	CALL EAX	
00A13A7B	. 83F8	CMP EAX,10	

Nos detuvimos en **00A13A69**; y aquí cae de maravilla el tuto anterior que compartí con todos ustedes, [1686](#), porque vamos a hacer lo mismo, desde aquí seguimos traceando con <F8> mientras se carga toda la consola y el programa se ejecuta, y así sabremos en que **CALL** se termina de cargar el programa.

Address | Hex | Comment

00D53195	B9 8060DB00	MOV ECX,Ret0.00DB6090
00D5319A	E8 51140000	CALL Ret0.00D545F0
00D5319F	8B00 0874DB00	MOV ECX,DWORD PTR DS:[0874DB00]
00D531A5	33FF	XOR EDI,EDI
00D531A7	85C9	TEST ECX,ECX
00D531A9	0F84 9E000000	JE Ret0.00D5324C
00D531AF	3BF9	CMP EDI,ECX
00D531B1	76 3A	JBE SHORT Ret0.00D531ED
00D531B3	68 6E0C0000	PUSH 0C6E
00D531B8	68 E81DB000	PUSH Ret0.00DB1BE8
00D531BD	68 AC1CDB00	PUSH Ret0.00DB1CA0
00D531C2	E8 89590000	CALL Ret0.00D58B50
00D531C7	83C4 0C	ADD ESP,0C
00D531CA	57	PUSH EDI
00D531CB	68 6E0C0000	PUSH 0C6E
00D531D0	68 E81DB000	PUSH Ret0.00DB1BE8
00D531D5	68 E81CDB00	PUSH Ret0.00DB1CE8
00D531DA	68 A81DDB00	PUSH Ret0.00DB1DA8
00D531DF	E8 3CD80000	CALL Ret0.00D60A20

CAUsers\LUISEFCAB\Doc...  
CLS Team 2018 ! Dev Bym24v  
Enter Password:

Ejecutamos unas cuantas instrucciones hasta que llegamos a **00D5319A CALL Ret0.00D545F0**, que es el **CALL** donde se carga la **Consola**, y paso seguido colocaremos un <BREAKPOINT> en la instrucción siguiente **00D5319F** para detenernos cuando hallamos ingresado nuestro **PASSWORD**, "MUYDIFICIL" y lo probemos.

Address | Hex | Comment

00D5319F	8B00 0874DB00	MOV ECX,DWORD PTR DS:[0874DB00]
00D531A5	33FF	XOR EDI,EDI
00D531A7	85C9	TEST ECX,ECX
00D531A9	0F84 9E000000	JE Ret0.00D5324C
00D531AF	3BF9	CMP EDI,ECX
00D531B1	76 3A	JBE SHORT Ret0.00D531ED
00D531B3	68 6E0C0000	PUSH 0C6E
00D531B8	68 E81DB000	PUSH Ret0.00DB1BE8
00D531BD	68 AC1CDB00	PUSH Ret0.00DB1CA0
00D531C2	E8 89590000	CALL Ret0.00D58B50
00D531C7	83C4 0C	ADD ESP,0C
00D531CA	57	PUSH EDI
00D531CB	68 6E0C0000	PUSH 0C6E
00D531D0	68 E81DB000	PUSH Ret0.00DB1BE8
00D531D5	68 E81CDB00	PUSH Ret0.00DB1CE8
00D531DA	68 A81DDB00	PUSH Ret0.00DB1DA8
00D531DF	E8 3CD80000	CALL Ret0.00D60A20
00D531E4	8B00 0874DB00	MOV ECX,DWORD PTR DS:[0874DB00]
00D531EA	83C4 14	ADD ESP,14
00D531ED	833D 0C74DB00	CMP DWORD PTR DS:[0C74DB00],10
00D531F4	B8 F873DB00	MOV EAX,Ret0.00DB73F8
00D531F9	0F4505 F873DB00	CMOVB EAX,DWORD PTR DS:[0B73F8]
00D53200	0FB0438	MOVSX EAX,BYTE PTR DS:[EAX+EDI]
00D53204	99	CDQ
00D53205	2BC2	SUB EAX,EDX

Arg3 = 00000C6E  
Arg2 = 00DB1BE8  
Arg1 = 00DB1CE8  
Ret0.00F20A20

ASCII "MUYDIFICIL"

## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISFECAB]

Ya aparece nuestro "MUYDIFICIL" y lo vemos en la dirección **00D531F4**. Hay un par de **CALL** antes, que en realidad no hacen nada con respecto a nuestro **PASSWORD**, yo no noté nada especial ahí, la verdad no tengo idea qué hacen, son basura para mortificarnos. Lo interesante empieza en la dirección **00D531ED**. Tracemos con <F8> pasa pasar esos **CALL** que solo estorban hasta llegar a **00D531ED**.

00D531ED	>	83D0 0C	CMP DWORD PTR DS:[DB740C],10	
00D531F4	.	B8 F873	MOV EAX,Ret0.00D873F8	ASCII "MUYDIFICIL"
00D531F9	.	0F4305	CMOVB EAX,DWORD PTR DS:[DB73F8]	
00D53200	.	0FB8043	MOVSX EAX,BYTE PTR DS:[EAX+EDI]	
00D53204	.	99	CDQ	
00D53205	.	2BC2	SUB EAX,EDX	
00D53207	.	D1F8	SAR EAX,1	
00D53209	.	A3 1074	MOV DWORD PTR DS:[DB7410],EAX	
00D5320E	.	83F8 21	CMF EAX,21	
00D53211	✓	7F 07	JG SHORT Ret0.00D5321F	Salta si es mayor
00D53213	.	03C0	ADD EAX,EAX	
00D53215	.	A3 1074	MOV DWORD PTR DS:[DB7410],EAX	
00D5321A	>	83F8 7D	CMF EAX,7D	
00D5321D	✓	7C 0A	JL SHORT Ret0.00D53225	Salta si es menor
00D5321F	.	99	CDQ	
00D53220	.	2BC2	SUB EAX,EDX	
00D53222	.	D1F8	SAR EAX,1	
00D53224	.	A3 1074	MOV DWORD PTR DS:[DB7410],EAX	
00D53229	>	8BF1	MOV ESI,ECX	Nuevo longitud String
00D5322B	.	0FAFF0	IMUL ESI,EAX	Multiplica Caracter x longitud
00D5322E	.	B8 C0CC	MOV EAX,CCCCCCC	
00D53233	.	F7E1	MUL ECX	Multiplica por longitud
00D53235	.	C1EA 02	SHR EDX,2	Resto de la multiplicacion
00D53238	.	8BCA	MOV ECX,EDX	
00D5323A	.	D3EE	SHR ESI,CL	
00D5323C	.	56	PUSH ESI	
00D5323D	.	E8 CEFB	CALL Ret0.00D52E10	ESI ES EL VALOR A UTILIZAR PARA ENTRAR AL SWITCH QUE HAY EN EL CALL Ret0.00D52E10
00D53242	.	8B0D 08	MOV ECX,DWORD PTR DS:[DB7408]	
00D53248	.	47	INC EDI	
00D53249	.	3BF9	CMF EDI,ECX	
00D5324B	✓	72 A0	JG SHORT Ret0.00D531ED	

Ese **LOOP** es nuestra **ZONA CALIENTE**, ahí obtendremos la **FLAG** cuando ingresemos un **PASSWORD** correcto. Lo que se hace aquí es lo siguiente: Se tomará carácter por carácter y se le aplican determinadas operaciones para obtener en el registro **ESI** un valor el cual será utilizado en **00D5323D CALL Ret0.00D52E10**, en donde tenemos un **Switch** o **Select Case**, pero antes de entrar a ese **CALL Ret0.00D52E10**, la tarea es que entiendan que operaciones se realizan aquí en ese **LOOP** porque debemos entenderlo bien para poder hallar el **PASSWORD** correcto con lo que ocurre ahí. Lo más importante es que noten que la longitud de nuestro **PASSWORD** es utilizada para validar el carácter. Entonces, puede que un mismo carácter sirva para una determinada longitud pero no para otra. La longitud del **PASSWORD** es la que determina todo, es la mamá de los pollitos, pero claro aún no sabemos cuál es la longitud ideal del **PASSWORD CORRECTO**.

Estoy explicando de manera general, si apenas empiezas a leer tutoriales de cracking puede que este no sea muy claro pero veremos si explicamos el primer carácter en el **LOOP** para dejarlo más claro.



## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISEECAB]

```

000531EA . 83C4 14 ADD ESP,14
000531ED > 833D 0C CMP DWORD PTR DS:[DB740C],10
000531F4 . B8 F873 MOV EAX,Ret0.00DB73F8 ASCII "MUYDIFICIL"
000531F9 . 0F4305 CMOVNB EAX,DWORD PTR DS:[DB73F8]
00053200 . 0FBE043 MOVSB EAX,BYTE PTR DS:[EAX+EDI]
00053204 . 99 CDB
00053205 . 2BC2 SUB EAX,EDX
00053207 . D1F8 SAR EAX,1
00053209 . A3 1074 MOV DWORD PTR DS:[DB7410],EAX
0005320E . 83F8 21 CMP EAX,21
00053211 . 7F 07 JG SHORT Ret0.00D5321F Salta si es mayor
00053213 . 03C0 ADD EAX,EAX
00053215 . A3 1074 MOV DWORD PTR DS:[DB7410],EAX
0005321A > 83F8 7D CMP EAX,7D
0005321D . 7C 0F JL SHORT Ret0.00D53229 Salta si es menor

```

Condition is FALSE  
DS:[00DB73F8]=4459554D  
EAX=00DB73F8 (Ret0.00DB73F8), ASCII "MUYDIFICIL"

Siguiendo la **FLECHA VERDE** vemos que el **PASSWORD** se mueva al registro **EAX** para más adelante tomar el primer carácter con **MOVSB**. Lo que más me interesa es mostrar lo que muestran las **FLECHAS ROJAS**, es la instrucción **CMOVNB** que lo que hace es mover a un registro un valor o dato, pero en función de que **FLAG-C=0**, y si vemos nosotros tenemos **FLAG-C=1** y con eso no moverá nada, vemos en las aclaraciones que **Condition is FALSE**. Si cambiáramos la **FLAG-C=0**.

```

000531EA . 83C4 14 ADD ESP,14
000531ED > 833D 0C CMP DWORD PTR DS:[DB740C],10
000531F4 . B8 F873 MOV EAX,Ret0.00DB73F8 ASCII "MUYDIFICIL"
000531F9 . 0F4305 CMOVNB EAX,DWORD PTR DS:[DB73F8]
00053200 . 0FBE043 MOVSB EAX,BYTE PTR DS:[EAX+EDI]
00053204 . 99 CDB
00053205 . 2BC2 SUB EAX,EDX
00053207 . D1F8 SAR EAX,1
00053209 . A3 1074 MOV DWORD PTR DS:[DB7410],EAX
0005320E . 83F8 21 CMP EAX,21
00053211 . 7F 07 JG SHORT Ret0.00D5321F Salta si es mayor
00053213 . 03C0 ADD EAX,EAX

```

Condition is TRUE  
DS:[00DB73F8]=4459554D  
EAX=00DB73F8 (Ret0.00DB73F8), ASCII "MUYDIFICIL"

Al poner **FLAG-C=0** vemos en las aclaraciones que **Condition is TRUE** y ahí sí movería algo. En nuestro caso esto nunca ocurrirá porque **CMP DWORD PTR DS:[DB740C],10** siempre activará la **FLAG-C=1**. Solo quería dar a conocer esa instrucción **CMOVNB** que no conocía y puede que más adelante nos la encontremos en otra aplicación y ahí si mueva algo. Sigamos, tomará el primer carácter.

Address	Hex dump	Disassembly	Comment	Registers (FPU)
000531ED	> 833D 0C	CMP DWORD PTR DS:[DB740C],10		EAX 0000004D
000531F4	. B8 F873	MOV EAX,Ret0.00DB73F8	ASCII "MUYDIFICIL"	ECX 0000000A
000531F9	. 0F4305	CMOVNB EAX,DWORD PTR DS:[DB73F8]		EDX 00DB50B0
00053200	. 0FBE043	MOVSB EAX,BYTE PTR DS:[EAX+EDI]		EBX 7FFD0000
00053204	. 99	CDB		ESP 0023F9D4
00053205	. 2BC2	SUB EAX,EDX		EBP 0023FCF8
00053207	. D1F8	SAR EAX,1		ESI 00000000

Tenemos en **EAX=4D** que será la letra **"M"**. Voy a llegar a donde se guarda nuestro valor de interés de **ESI** en el **STACK** porque explicar cada cálculo no se justifica, son cálculos que usted mismo mi apreciado lector puede entender y deducir.

## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISEECAB]

Address	Hex dump	Disassembly	Comment	Registers (FPU)
00D53235	. C1EA 02	SHR EDX,2	Resto de la multip	EAX 00000002
00D53238	. 8BCA	MOV ECX,EDX		ECX 00000002
00D5323A	. D3EE	SHR ESI,CL		EDX 00000002
00D5323C	. 56	PUSH ESI	Arg1 = 0000005F	EBX 7FFDD000
00D5323D	. E8 CEFB	CALL Ret0.00D52E10	Ret0.00F12E10	ESP 0023F9D4
00D53242	. 8B0D 08	MOV ECX,DWORD PTR DS:[DB74008]		EBP 0023FCF8
00D53248	. 47	INC EDI		ESI 0000005F
00D53249	. 3BF9	CMP EDI,ECX		EDI 00000000
00D5324B	. 72 A0	JB SHORT Ret0.00D531ED		EIP 00D5323C

Con la longitud de mi **PASSWORD**, "MUYDIFICIL" que es **0x0A** (10) y **0x4D** ("M") obtengo **0x5F**, podemos decir que ese valor determina si vamos por buen camino o no. Ahora si entremos al **CALL Ret0.00D52E10**.

Address	Hex dump	Disassembly	Comment	Registers (FPU)
00D52E10	. 55	PUSH EBP		EAX 00000002
00D52E11	. EC	MOV EBP,ESP		ECX 00000002
00D52E13	. 56	PUSH ECX		EDX 00000002
00D52E14	. B645	MOVZX EAX,BYTE PTR SS:[EBP+0]		EBX 7FFDD000
00D52E18	. 15 F0	MOV EDI,DWORD PTR DS:[DB73F0]		ESP 0023F9CC
00D52E1E	. C0 E4	ADD EAX,-1C	Switch (cases 1C..37)	EBP 0023FCF8
00D52E21	. F8 1B	CMP EAX,1B		ESI 0000005F
00D52E24	. 87 F6	JR Ret0.00D53120	-- Salta si es mayor	EDI 00000000
00D52E2A	. B680	MOVZX EAX,BYTE PTR DS:[EAX+D53160]		EIP 00D52E10 R
00D52E31	. 2485	JMP DWORD PTR DS:[EAX*4+D53120]		C 0 ES 0023 3
00D52E38	. 42 04	MOV EAX,DWORD PTR DS:[EDX+4]	1; Case 1C of switch 00D52E1E	P 1 CS 001B 3
00D52E3B	. C0	TEST EAX,EAX		A 1 SS 0023 3
00D52E3D	. 18	JNZ SHORT Ret0.00D52E57		Z 0 DS 0023 3
00D52E3F	. 42 08	MOV EAX,DWORD PTR DS:[EDX+8]		S 0 FS 003B 3
00D52E42	. 4410	MOV BYTE PTR DS:[EAX+EDX+C],23		
00D530D5	. 1F	JMP SHORT Ret0.00D530F6		
00D530D7	. 7A 04	CMP DWORD PTR DS:[EDX+4],12	12; Case 2D of switch 00D52E1E	
00D530DB	. 04	JMP SHORT Ret0.00D530E1		
00D530DD	. 7A 04	CMP DWORD PTR DS:[EDX+4],15	13; Case 31 of switch 00D52E1E	
00D530E1	. 3D	JNZ SHORT Ret0.00D53120		
00D530E3	. 42 08	MOV EAX,DWORD PTR DS:[EDX+8]		
00D530E6	. 4410	MOV BYTE PTR DS:[EAX+EDX+C],2F		
00D530EB	. 42 08	INC DWORD PTR DS:[EDX+8]		
00D530EE	. 42 08	MOV EAX,DWORD PTR DS:[EDX+8]		
00D530F1	. 4410	MOV BYTE PTR DS:[EAX+EDX+C],30		
00D530F6	. 42 08	INC DWORD PTR DS:[EDX+8]		
00D530F9	. 42 08	MOV EAX,DWORD PTR DS:[EDX+8]		
00D530FC	. 0C10	LEA ECX,DWORD PTR DS:[EAX+EDX]		
00D530FF	. B6441	MOVZX EAX,BYTE PTR DS:[EAX+EDX+B]		
00D53104	. 41 0A	ADD AL,BYTE PTR DS:[ECX+A]		
00D53107	. 41 0D	MOV BYTE PTR DS:[ECX+D],AL		
00D5310A	. 0A	MOV ECX,DWORD PTR DS:[EDX]		
00D5310C	. 42 08	MOV DWORD PTR DS:[EDX+8],0		
00D53113	. B642	MOVZX EAX,BYTE PTR DS:[EDX+F]		
00D53117	. 4411	MOV BYTE PTR DS:[ECX+EDX+7B],AL		
00D5311B	. 2	INC DWORD PTR DS:[EDX]		
00D5311D	. 42 04	INC DWORD PTR DS:[EDX+4]		
00D53120	. BE5	MOV ESP,EBP	Default case of switch 00D52E1E	
00D53122	. 5D	POP EBP		
00D53123	. C2 0400	RETN 4		

Es un larguero, he cortado una parte para poder mostrar el inicio y final del **Switch**. Podemos darnos cuenta que hay **13** entradas al **Switch** o **13 Case** que se toman y que para ser tomados dependen del valor de nuestro **ESI**. Ojeamos el inicio para ver que tenemos antes de llegar al **00D52E14 JA Ret0.00D53120**.



# Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISEECAB]

Address	Hex	Dump	Disassembly	Registers (FPU)
00052E10	55		PUSH EBP	EAX 0000005F
00052E11	8BEC		MOV EBP,ESP	ECX 00000002
00052E13	51		PUSH ECX	EDX 0029A020
00052E14	0FB645		MOVZX EAX, BYTE PTR SS:[EBP+8]	
00052E18	8B15 F0		MOV EDX, DWORD PTR DS:[0023F9F0]	
00052E1E	83C0 E4		ADD EAX, -1C	
00052E21	83F8 1B		CMP EAX, 1B	
00052E24	0F87 F6		JA Ret0.00D53120	

A NUESTRO ESI, 0x5F LE SUMA -1C. EN REALIDAD LE RESTA.

LUEGO COMPARA CON 1B. JA SE ACTIVA SI EAX MENOR O IGUAL. NO DEBE SALTAR.

EBP-4	00000002
EBP ==>	0023FCF8
EBP+4	00D53242
EBP+8	0000005F
EBP+C	0023F9DC
EBP+10	0023F9F8
EBP+14	C0000005
EBP+18	00000000

RETURN to Ret0.00D53242 from Ret0.00D52E10

Nuestro valor **ESI** es tomado del **STACK** y cargado en **EAX=5F**, luego le suma **0x-1C** que es todo lo contrario, en realidad le resta, y ese nuevo valor lo comparamos con **0x1B** y de acuerdo a esa comparación se activará o no el salto **00D52E14 JA Ret0.00D53120**. Empecemos a tracear con <F7> o <F8> hasta llegar al salto.

Address	Hex	Dump	Disassembly	Registers (FPU)
00052E10	55		PUSH EBP	EAX 00000043
00052E11	8BEC		MOV EBP,ESP	ECX 00000002
00052E13	51		PUSH ECX	EDX 0029A020
00052E14	0FB645		MOVZX EAX, BYTE PTR SS:[EBP+8]	EBX 7FFD0000
00052E18	8B15 F0		MOV EDX, DWORD PTR DS:[0023F9F0]	ESP 0023F9C4
00052E1E	83C0 E4		ADD EAX, -1C	EBP 0023F9C8
00052E21	83F8 1B		CMP EAX, 1B	ESI 0000005F
00052E24	0F87 F6		JA Ret0.00D53120	EDI 00000000
00052E2A	0FB680		MOVZX EAX, BYTE PTR DS:[EAX+053160]	EIP 00D52E24
00052E31	FF2485		JMP DWORD PTR DS:[EAX*4+053120]	

Switch (cases 1C..37)

-- Salta si es mayor

Nuestro valor de entrada en **ESI**, **0x5F**, terminó siendo **0x43** en **EAX=43**, y debido a eso el salto **JA** se activa. Miremos las condiciones de ese salto.

JA	Jump if above	unsigned	CF = 0 and ZF = 0
JNBE	Jump if not below or equal		

El salto se activará si **EAX > 1B** y como vemos **EAX=43**, entonces tomaremos el salto y no entraremos al **Switch**, lo cual es una muy mala noticia, cosa que no debería pasar, así que nuestro **PASSWORD** no sirve.

Hasta aquí se puede decir que no tuve dificultad en analizar y entender lo que hace el <Ret0> para comprobar el **PASSWORD**. Con lo que hemos averiguado hasta este momento podemos plantear la solución, y es buscar qué caracteres sirven para diferentes longitudes que cumplan la condición del **00D52E14 JA Ret0.00D53120**; y eso fue lo que más me demoró porque no solo es evitar el salto si no entender que sucedía dentro del **Switch** y cuáles eran los valores para poder tomar los **13 Cases** que hay dentro del **Switch**.

Address	Hex	Dump	Disassembly	Registers (FPU)
00052E14	0FB645		MOVZX EAX, BYTE PTR SS:[EBP+8]	
00052E18	8B15 F0		MOV EDX, DWORD PTR DS:[0023F9F0]	
00052E1E	83C0 E4		ADD EAX, -1C	
00052E21	83F8 1B		CMP EAX, 1B	
00052E24	0F87 F6		JA Ret0.00D53120	
00052E2A	0FB680		MOVZX EAX, BYTE PTR DS:[EAX+053160]	
00052E31	FF2485		JMP DWORD PTR DS:[EAX*4+053120]	
00052E38	8B42 04		MOV EAX, DWORD PTR DS:[EDX+4]	
00052E3B	85C0		TEST EAX, EAX	

VALOR DE EAX.

Switch (cases 1C..37)

-- Salta si es mayor

1; Case 1C of switch 00D52E1E

## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISEECAB]

Por fortuna, en la columna **<Comment>** podemos saber qué valor debe tener **EAX** para no saltar y de paso poder tomar los **13 Case**. Por ejemplo, para entrar al **Primer Case**, **EAX** debe valer **0x1C**.

Ya con eso conocemos todo lo necesario para buscar posibles caracteres que nos sirvan para entrar al **Switch**. La búsqueda lo hacemos mediante **FUERZA BRUTA**, pero claro, para llegar a esta solución no fue así a la primera como aquí, primero hice muchas pruebas hasta saberme de memoria la **ZONA CALIENTE** y entrando al **Switch** a la fuerza, cambiando una de las dos **FLAGS** que trabajan en conjunto para activar el salto **JA**. Para que **JA** se active debe ser **FLAG-C=0** y **FLA-Z=0**, con cambiar alguna de esas **FLAGS** a **1**, no tomamos el salto y entramos al **Switch**.

Address	Hex dump	Disassembly	Comment
00052E1E	83C0 E4	ADD EAX,-1C	Switch (cases 1C..37)
00052E21	83F8 1B	CMP EAX,1B	
00052E24	0F87 F6	JA Ret0.00053120	-- Salta si es mayor
00052E2A	0FB680	MOVZX EAX, BYTE PTR DS:[EAX+D53160]	
00052E31	FF2485	JMP DWORD PTR DS:[EAX+4+D53128]	
00052E38	8B42 04	MOV EAX, DWORD PTR DS:[EDX+4]	1; Case 1C of switch 00052E1E
00052E3B	85C0	TEST EAX,EAX	
00052E3D	75 18	JNZ SHORT Ret0.00052E57	
00052E3F	8B42 08	MOV EAX, DWORD PTR DS:[EDX+8]	
00052E42	C64410	MOV BYTE PTR DS:[EAX+EDX+C],23	
00052E47	FF42 08	INC DWORD PTR DS:[EDX+8]	
00052E4A	8B42 08	MOV EAX, DWORD PTR DS:[EDX+8]	

Registers (FPU):  
EAX 00000043  
ECX 00000002  
EDX 0029A020  
EBX 7FFD0000  
ESP 0023F9C4  
EBP 0023F9C8  
ESI 0000005F  
EDI 00000000  
EIP 00052E2A Ret0.0  
C 1 ES 0023 32bit  
P 1 CS 001B 32bit

Cambiamos **FLAG-C=1** para evitar tomar el salto y quedamos parados en **00052E2A MOVZX EAX, BYTE PTR DS:[EAX+9D3160]**, en la imagen de arriba la dirección es **D53160** y en la imagen de abajo es otra, **9D3160**, y se debe a que tuve que cerrar todo e iniciar de nuevo, y al hacer eso se cargan diferentes direcciones, cosa que no cambia nada porque lo importante es seguir esa dirección y listo. Aquí es donde el valor de **EAX** determina el salto a un determinado **Case**. Miremos en el **DUMP** la dirección **9D3160** para ver los valores que tomará ahora **EAX** cuando se haga el **MOVZX**, los cuales darán el valor del salto.

Address	Hex dump	ASCII
009D3160	00 00 00 00 00 00 01 00 00 00 00 02 03 04 05 06	.....0....0+4
009D3170	07 08 00 00 09 0A 0B 00 00 00 00 0C 0D 0E 0F 10	..8...9AB...CDEF
009D3180	11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20	..123456789ABCDEF

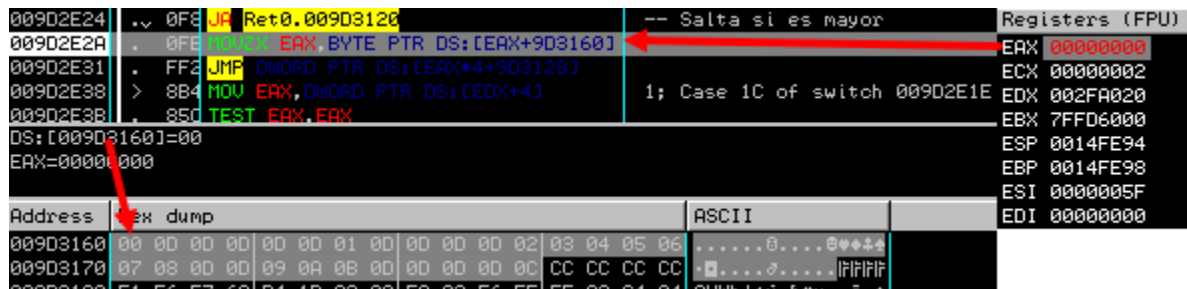
En el **DUMP** podemos ver que **RESALTADO EN VERDE** tenemos los **13** valores que no llevarán a los **Case**. También tenemos entre esos valores el **0x0D** que vendría siendo el **Case Default**. Para poder tomar el **Primer Case**, **EAX** debe ser **0x00**, pero como venimos mal desde un inicio y entramos al **Switch** haciendo trampa, entonces en **EAX** tenemos otro valor, **0x43**.

Address	Hex dump	Disassembly	Comment
009D2E24	0F87 F6	JA Ret0.009D3120	-- Salta si es mayor
009D2E2A	0FB680	MOVZX EAX, BYTE PTR DS:[EAX+9D3160]	
009D2E31	FF2485	JMP DWORD PTR DS:[EAX+4+9D3128]	
009D2E38	8B42 04	MOV EAX, DWORD PTR DS:[EDX+4]	1; Case 1C of switch 009D2E1E
009D2E3B	85C0	TEST EAX,EAX	

Registers (FPU):  
EAX 00000043  
ECX 00000002  
EDX 002FA020  
EBX 7FFD6000  
ESP 0014FE94

## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISEFECAB]

Pues sigamos haciendo trampa, pongamos el registro **EAX=0** que sería como si tuviéramos un carácter válido el cual tomaría el **Primer Case**.



```
009D2E24 .< 0FB JN Ret0.009D3126
009D2E2A . 0FB MOVZX EAX, BYTE PTR DS:[EAX+9D3160]
009D2E31 . FF2 JMP DWORD PTR DS:[EAX*4+9D3128]
009D2E38 > 8B4 MOV EAX, DWORD PTR DS:[EDX+4]
009D2E3B . 850 TEST EAX, EAX
DS:[009D3160]=00
EAX=00000000
Registers (FPU)
EAX 00000000
ECX 00000002
EDX 002FA020
EBX 7FFD6000
ESP 0014FE94
EBP 0014FE98
ESI 0000005F
EDI 00000000
Address Hex dump ASCII
009D3160 00 0D 0D 0D 0D 0D 01 0D 0D 0D 0D 02 03 04 05 06 .....0....00+2+
009D3170 07 08 0D 0D 09 0A 0B 0D 0D 0D 0D 0C CC CC CC CC +0....0....|f|f|f|f|
009D3180 51 56 57 60 04 10 03 08 50 03 56 5F 5F 03 04 04 00000000000000000000000000000000
```

Vemos que con la instrucción **00D52E2A MOVZX EAX, BYTE PTR DS:[EAX+9D3160]** a **EAX** se le pasará **0x00** y de esa forma poder saltar al lugar correcto. Miremos como el salto va a donde queremos.



```
009D2E24 .< 0FB JN Ret0.009D3126
009D2E2A . 0FB MOVZX EAX, BYTE PTR DS:[EAX+9D3160]
009D2E31 < FF2 JMP DWORD PTR DS:[EAX*4+9D3128] Ret0.009D2E38
009D2E38 > 8B4 MOV EAX, DWORD PTR DS:[EDX+4] 1: Case 1C of switch 009D2E1E
009D2E3B . 850 TEST EAX, EAX
009D2E3D < 75 JNZ SHORT Ret0.009D2E57
009D2E3F > 8B4 MOV EAX, DWORD PTR DS:[EDX+8]
009D2E42 . C64 MOV BYTE PTR DS:[EAX+EDX+C1], 23
009D2E47 . FF4 INC DWORD PTR DS:[EDX+8]
009D2E4A . 8B4 MOV EAX, DWORD PTR DS:[EDX+8]
009D2E4D . C64 MOV BYTE PTR DS:[EAX+EDX+C1], 23
009D2E52 .< E9 JMP Ret0.009D30F6
```

El salto nos envía al **Primer Case**. Debemos seguir traceando a ver qué podemos descubrir. De esa forma, traciando sin parar, cambiando las **FLAGS** y los valores en **EAX** fue que fui resolviendo el reto y llegando a la solución. Miremos esta instrucción, **009D2E38 MOV EAX, DWORD PTR DS:[EDX+4]**, ahora no dice mucho pero lo que hace es pasar a **EAX** el valor de un contador, y que es muy importante porque ese contador determina el orden para ir generando nuestra **FLAG**. Sigamos traciando hasta llegar al siguiente salto.



```
009D2E38 > 8B4 MOV EAX, DWORD PTR DS:[EDX+4] 1: Case 1C of switch 009D2E1E
009D2E3B . 850 TEST EAX, EAX
009D2E3D < 75 JNZ SHORT Ret0.009D2E57
009D2E3F > 8B4 MOV EAX, DWORD PTR DS:[EDX+8]
009D2E42 . C64 MOV BYTE PTR DS:[EAX+EDX+C1], 23
009D2E47 . FF4 INC DWORD PTR DS:[EDX+8]
009D2E4A . 8B4 MOV EAX, DWORD PTR DS:[EDX+8]
009D2E4D . C64 MOV BYTE PTR DS:[EAX+EDX+C1], 23
009D2E52 .< E9 JMP Ret0.009D30F6
Registers (FPU)
EAX 00000000
ECX 00000002
EDX 002FA020
EBX 7FFD6000
ESP 0014FE94
EBP 0014FE98
ESI 0000005F
EDI 00000000
```

Nuestro contador empieza en **0x00** y como vemos en **009D2E3D JNZ SHORT Ret0.009D2E57** no tomamos el salto si no que entramos a generar el primer carácter de nuestra **FLAG** en el **RECUADRO VERDE**, lo que hará es guardar esos dos valores en memoria que en este caso son iguales, un **0x23**. Lleguemos al salto **002E2E52 JMP Ret0.002E30F6** para ver los valores guardados en el **DUMP**.

# Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISFECAB]

```

002E2E3D < 75 18 JNZ SHORT Ret0.002E2E57
002E2E3F > 8B42 08 MOV EAX,DWORD PTR DS:[EDX+8]
002E2E42 . C64410 0C 23 MOV BYTE PTR DS:[EAX+EDX+C],23
002E2E47 . FF42 08 INC DWORD PTR DS:[EDX+8]
002E2E4A . 8B42 08 MOV EAX,DWORD PTR DS:[EDX+8]
002E2E4D . C64410 0C 23 MOV BYTE PTR DS:[EAX+EDX+C],23
002E2E52 < E9 9F020000 JMP Ret0.002E30F6
002E2E57 > 83F8 01 CMP EAX,1
002E2E59 . 75 18 JNZ SHORT Ret0.002E2E74
002E30F6=Ret0.002E30F6

```

Address	Hex dump	ASCII
003AA028	01 00 00 00 23 23 00 00 00 00 00 00 00 00 00 00	0...##...
003AA038	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Ahí los movió y con esos se origina el primer carácter de la **FLAG**, claro que hasta aquí no sabemos eso, pero si sumamos  $0x23+0x23=0x46$ , que en **ASCII** es "F". Saltemos **002E2E52 JMP Ret0.002E30F6** para llegar a la parte donde ocurre la suma de los dos valores para originar el carácter de la **FLAG** y aumentar nuestro contador.

```

002E30F6 < FF42 08 INC DWORD PTR DS:[EDX+8]
002E30F9 . 8B42 08 MOV EAX,DWORD PTR DS:[EDX+8]
002E30FC . 8D0C10 LEA ECX,DWORD PTR DS:[EAX+EDX]
002E30FF . 0FB64410 0B MOVZX EAX,BYTE PTR DS:[EAX+EDX+8]
002E3104 . 0241 0A ADD AL,BYTE PTR DS:[ECX+8]
002E3107 . 8B41 0D MOV BYTE PTR DS:[ECX+D],AL
002E310A . 8B0A MOV ECX,DWORD PTR DS:[EDX]
002E310C . C742 08 0000 MOV DWORD PTR DS:[EDX+8],0
002E3113 . 0FB642 0F MOVZX EAX,BYTE PTR DS:[EDX+F]
002E3117 . 8B4411 70 MOV BYTE PTR DS:[ECX+EDX+70],AL
002E311B . FF02 INC DWORD PTR DS:[EDX]
002E311D . FF42 04 INC DWORD PTR DS:[EDX+4]
002E3120 > 8BE5 MOV ESP,EBP
002E3122 . 5D POP EBP
002E3123 . C2 0400 RETN 4

```

Arriba vemos el procedimiento que hará la suma para sacar el carácter y aumentar nuestro contador. Lleguemos hasta el **Retn 4** y miremos en el **DUMP** cómo va haciendo la tarea.

```

00BA30E6 . C64410 0C 2F MOV BYTE PTR DS:[EAX+EDX+C],2F
00BA30EB . FF42 08 INC DWORD PTR DS:[EDX+8]
00BA30EE . 8B42 08 MOV EAX,DWORD PTR DS:[EDX+8]
00BA30F1 . C64410 0C 30 MOV BYTE PTR DS:[EAX+EDX+C],30
00BA30F6 > FF42 08 INC DWORD PTR DS:[EDX+8]
00BA30F9 . 8B42 08 MOV EAX,DWORD PTR DS:[EDX+8]
00BA30FC . 8D0C10 LEA ECX,DWORD PTR DS:[EAX+EDX]
00BA30FF . 0FB64410 0B MOVZX EAX,BYTE PTR DS:[EAX+EDX+8]
00BA3104 . 0241 0A ADD AL,BYTE PTR DS:[ECX+8]
00BA3107 . 8B41 0D MOV BYTE PTR DS:[ECX+D],AL
00BA310A . 8B0A MOV ECX,DWORD PTR DS:[EDX]
00BA310C . C742 08 0000 MOV DWORD PTR DS:[EDX+8],0
00BA3113 . 0FB642 0F MOVZX EAX,BYTE PTR DS:[EDX+F]
00BA3117 . 8B4411 70 MOV BYTE PTR DS:[ECX+EDX+70],AL
00BA311B . FF02 INC DWORD PTR DS:[EDX]
00BA311D . FF42 04 INC DWORD PTR DS:[EDX+4]
00BA3120 > 8BE5 MOV ESP,EBP
00BA3122 . 5D POP EBP
00BA3123 . C2 0400 RETN 4
00BA3126 . 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66

```

**NUESTRO CONTADOR.**

**LO DOS VALORES A SUMAR PARA HALLAR CARACTER**

**GUARDA LA SUMA (CARATER) PARA LUEGO MOVERLO PARA IR AMANDO LA FLAG.**

Address	Hex dump	ASCII
003AA020	01 00 00 00 01 00 00 00 00 00 00 00 23 23 00 46	0...#...F
003AA030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
003AA040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
003AA050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
003AA060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
003AA070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
003AA080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
003AA090	46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	F.....

**AQUÍ VA ARMANDO LA FLAG.**

## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISFECAB]

Se halla el carácter y se actualiza nuestro contador. Terminamos ese **CALL** que es el del **Switch** y regresamos a la **ZONA CALIENTE**, aunque en realidad nunca hemos salido de ella.

00BA323D	. E8 CEFBFFFF	CALL Ret0.00BA2E10	Ret0.00F12E10	Registers (MMX)
00BA3242	. 8B00 0B74C000	MOV ECX,DWORD PTR DS:[C074003]		EAX 00000046
00BA3248	. 47	INC EDI		ECX 0000000A
00BA3249	. 3BF9	CMP EDI,ECX		EDX 003AA020
00BA324B	. 72 A0	JNB SHORT Ret0.00BA31ED		EBX 7FFD9000
00BA324D	> 8B00 F073C000	MOV ECX,DWORD PTR DS:[C073F00]		ESP 0028F76C
				EBP 0028F788
				ESI 0000005F
				EDI 00000001

LONGITUD PASSWORD.

Sigue con el **LOOP**, nuestra longitud **0x0A** (10) y apenas vamos en **0x01**, así que se sigue probando carácter por carácter para ir armando la **FLAG**, pero claro se supone que es con un **PASSWORD VÁLIDO**. Miremos lo que hay después del **LOOP**.

009F3249	. 3BF9	CMP EDI,ECX		
009F324B	. 72 A0	JNB SHORT Ret0.009F31ED		
009F324D	> 8B00 F073A500	MOV ECX,DWORD PTR DS:[A573F00]		
009F3253	. 8B01	MOV EAX,DWORD PTR DS:[ECX]		
009F3255	. 85C0	TEST EAX,EAX		
009F3257	. 75 0B	JNZ SHORT Ret0.009F3264		
009F3259	. E8 22FFFFF	CALL Ret0.009F3180		
009F325E	. 5F	POP EDI		
009F325F	. 33C0	XOR EAX,EAX		
009F3261	. 5E	POP ESI		
009F3262	. 59	POP ECX		
009F3263	. C3	RETN		
009F3264	> 83F8 1B	CMP EAX,1B		
009F3267	. 75 F5	JNZ SHORT Ret0.009F325E		
009F3269	. 8D41 70	LEA EAX,DWORD PTR DS:[ECX+70]		
009F326C	. 50	PUSH EAX		
009F326D	. 68 E41BA500	PUSH Ret0.00A51BE4		
009F3272	. E8 19E5FFFF	CALL Ret0.009F1790		
009F3277	. 83C4 08	ADD ESP,8		
009F327A	. 6A 00	PUSH 0		

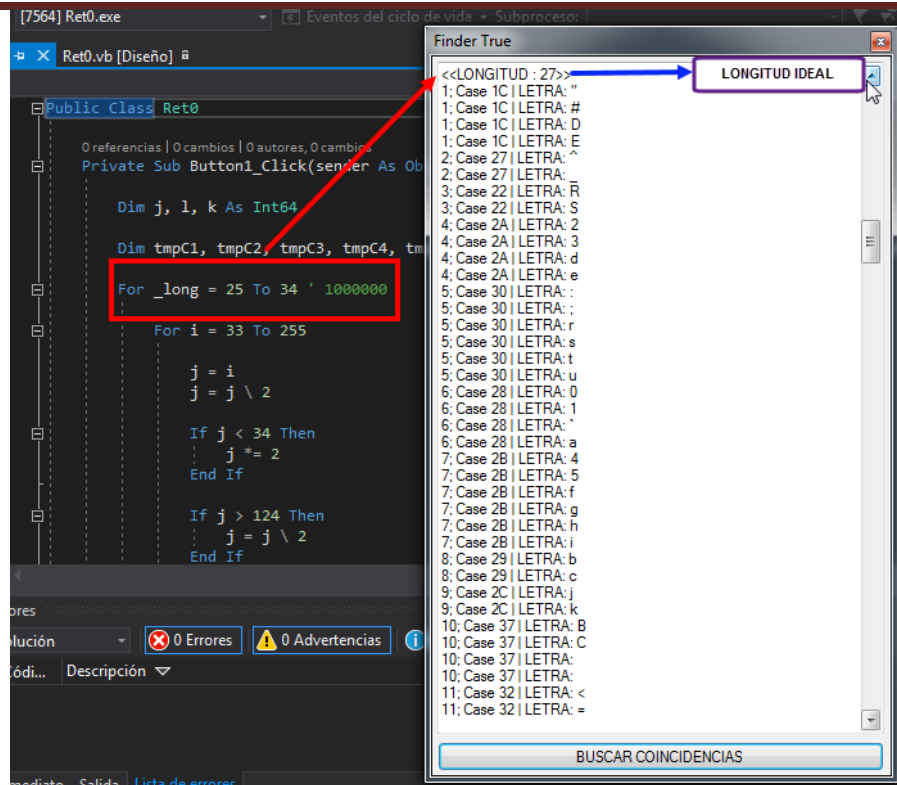
Arg2  
Arg1 = 00A51BE4 ASCII "%s0"  
Ret0.00391790

Tenemos dos comprobaciones más, la primera en **009F3255 TEST EAX,EAX**, esta revisa si nuestro contador del **Switch** ha aumentado, o sea, que si hemos entrado al **Switch** y pasado por un **Case** porque por ese camino es que aumenta nuestro contador. Con que hallamos entrado aunque sea una sola vez es suficiente para pasar la comprobación y tomar el salto, **009F3257 JNZ SHORT Ret0.009F3264**, para pasar a la segunda comprobación, **009F3264 CMP EAX,1B**. Esta segunda comprobación ya no se comprueba con el mismo contador, sino que ya nuestro contador debe valer **0x1B** para evitar activar el salto. Con pasar esa última comprobación podemos decir que hemos hallado la **FLAG** y vencido el <Ret0>. Esta última comprobación es la que nos da la longitud ideal de nuestro **PASSWORD** que sería es **0x1B** (27), y como ya sabemos que cada carácter de un **PASSWORD VÁLIDO** equivale a un carácter de la **FLAG** entonces nuestra **FLAG** también tiene la misma longitud, **0x1B** (27).

Creo que con todo lo explicado hasta aquí tengas suficientes herramientas para que puedas entender mucho mejor el <Ret0> y hacer tus propias pruebas y traceos para hallar la **FLAG**.

Dijimos que utilizando **FUERZA BRUTA** para encontrar los caracteres que servían para hallar nuestro **PASSWORD**.

# Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISFECAB]

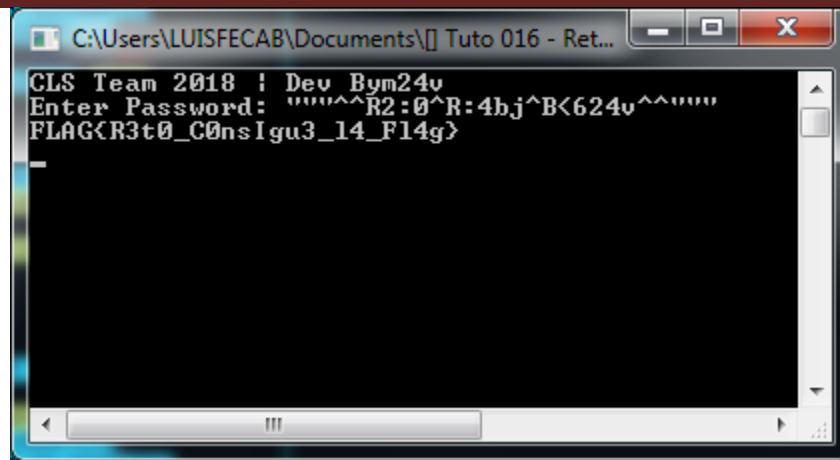


Yo hice mi aplicación en **VB.NET** para hallar qué caracteres me sirven para armar mi **PASSWORD**. Ahora sería escoger el orden de los **27** caracteres y eso viene dado por el mentado contador del **Switch**, recuerden que les dije que era muy importante. Bueno, yo les expliqué el **Primer Carácter** y cambiamos las **FLAGS** y valores en **EAX** para entrar al **Switch** y hallar el inicio correcto, ya con eso puedes hallar el resto del orden de entrada correcto a los demás **Case**. Aquí les dejo el **PASSWORD** que armé y además les muestro el orden correcto.

```
PASSWORD: """"^R2:0^R:4bj^B<624v^^""""
```

"=>Contador=0	[1; Case 1C]	==> 0x23+0x23 = 0x46 ("F")
"=>Contador=1	[1; Case 1C]	==> 0x26+0x26 = 0x4C ("L")
"=>Contador=2	[1; Case 1C]	==> 0x20+0x21 = 0x41 ("A")
^=>Contador=3	[2; Case 27]	==> 0x23+0x24 = 0x47 ("G")
^=>Contador=4	[2; Case 27]	==> 0x3D+0x3E = 0x7B ("{"
R=>Contador=5	[3; Case 22]	==> 0x29+0x29 = 0x52 ("R")
2=>Contador=6	[4; Case 2A]	==> 0x19+0x1A = 0x33 ("3")
:=>Contador=7	[5; Case 30]	==> 0x3A+0x3A = 0x74 ("t")
0=>Contador=8	[6; Case 28]	==> 0x18+0x18 = 0x30 ("0")
^=>Contador=9	[2; Case 27]	==> 0x2F+0x30 = 0x5F ("c")
R=>Contador=A	[3; Case 22]	==> 0x21+0x22 = 0x43 ("C")
:=>Contador=B	[5; Case 30]	==> 0x18+0x18 = 0x30 ("0")
4=>Contador=C	[7; Case 2B]	==> 0x37+0x37 = 0x6E ("n")
b=>Contador=D	[8; Case 29]	==> 0x39+0x3A = 0x73 ("s")
j=>Contador=E	[9; Case 2C]	==> 0x24+0x25 = 0x49 ("I")
^=>Contador=F	[2; Case 27]	==> 0x33+0x34 = 0x67 ("g")
B=>Contador=10	[10; Case 37]	==> 0x3A+0x3B = 0x75 ("u")
<=>Contador=11	[11; Case 32]	==> 0x19+0x1A = 0x33 ("3")
6=>Contador=12	[12; Case 2D]	==> 0x2F+0x30 = 0x5F ("c")
2=>Contador=13	[4; Case 2A]	==> 0x36+0x36 = 0x6C ("l")
4=>Contador=14	[7; Case 2B]	==> 0x1A+0x1A = 0x34 ("4")
v=>Contador=15	[13; Case 31]	==> 0x2F+0x30 = 0x5F ("c")
^=>Contador=16	[2; Case 27]	==> 0x23+0x23 = 0x46 ("F")
^=>Contador=17	[2; Case 27]	==> 0x36+0x36 = 0x6C ("l")
"=>Contador=18	[1; Case 1C]	==> 0x1A+0x1A = 0x34 ("4")
"=>Contador=19	[1; Case 1C]	==> 0x33+0x34 = 0x67 ("g")
"=>Contador=1A	[1; Case 1C]	==> 0x3E+0x3F = 0x7D ("}"))

Ahí lo tienen, nuestro contador y el orden de tomar los **Case** para armar nuestro **PASSWORD VÁLIDO** que nos permita obtener la **FLAG** que es: **"FLAG{R3t0\_C0nsIgu3\_14\_F14g}"**.



Probamos el **PASSWORD** y nos muestra la **FLAG**. Lo hemos logrado, vencimos el <Ret0>.



## PARA TERMINAR

Un buen reto que creo no es tan sencillo de abordar al principio, eso sentí yo, pero que con insistirle se puede hacer. Creo que si estamos empezando en **Cracking** se nos puede hacer un poco más difícil de vencer y puede que hasta el tuto sea medio enredado, es más creo que el tutorial no me quedó tan explicado o sencillo de entender como hubiera querido.

Siempre se pregunta que si debemos ser unos grandes programadores para entrarle al **Cracking** y la verdad es que no, lo importante es tener bases mínimas que te permitan poder programar ciertos procedimientos para realizar cálculos, que en casos como este reto <Ret0>, te permita resolverlo un poco más fácil porque al utilizar **FUERZA BRUTA** pudimos hallar los caracteres para armar el **PASSWORD** y resolver el reto completamente, digo esto porque hubiéramos podido hallar la **FLAG** forzando nosotros a tomar el camino correcto para llegar a la solución sin tener un **PASSWORD VÁLIDO**, claro que hallamos la **FLAG** pero no del modo que el <Ret0> lo pide.

He quedado muy contento con lograr este reto que me animé a hacerle un **KeyGen**. Como dije, con tener bases mínimas de programación te permitirán hacer tus propios **KeyGen** y cosas por el estilo, vuelvo y digo, de tener bases mínimas en programación porque eso tengo yo en programación, y si lo puedo hacer yo entonces cualquiera con esos mínimos conocimientos también.



Para agregarle música al **KeyGen** utilicé la librería **Bassmod** de [un4seen](#) para **.NET**. Sabía de su existencia y que es muy utilizada para reproducir los chiptunes como los **.xm** o **.mod** y como tenía un chiptune en **.xm** pues me decidí a utilizarla. Me tocó buscar información por **Intenet** de cómo se podía utilizar esa librería porque yo nunca había programado algo que tuviera que utilizar referencias de terceros pero por fortuna por Internet se obtiene mucha información del uso de esta. Esta librería te muestran una **NAG** si no estás registrado, pero te puedes registrar en

## Ret0 by Bym24v [OllyDBG v1.10][MVC++][CTF][LUISFECAB]

---

su **WEB** para obtener una Key y registrarla pero yo la parchee para evitarme la fatiga de registrarme.

Como ven, utilicé el **OllyDBG v1.10** que hace rato no lo utilizaba, es que me dio nostalgia por tenerlo tan abandonado. Este **OllyDBG** que he usado es el que uno va armando cuando hacemos el curso de **Maestro Ricardo**, [OLLY DESDE CERO](#).

Este reto es el primero de tres, codeados por **Bym24v**. Los otros dos restantes ni lo he visto, pero si este <Ret0> se me hizo exigente, no me quiero imaginar los otros dos.

Ha llegado el momento de despedirme de ti, mi querido lector, gracias por llegar hasta estas últimas líneas.

***@LUISFECAB***