Cheat Engine, conceptos básicos, VEH Debugger, Pointers, crear cheat GODMODE, Shared Codes, diseccionar Shared Codes, autoassemble, diseccionar estructuras, localizar estructuras dinámicas, AoB to Memory Data, Start VBS AoB pattern generator, comparar con WinMerge, TeleportHack

Cheat Engine

Nivel Avanzado

MadAntrax – elhacker.net

INTRODUCCIÓN

Muy buenas a todos.

Vuelvo con un nuevo tutorial para **Cheat Engine**, en éste caso vamos a profundizar al máximo y a explotar las funciones más avanzadas que nos ofrece **Cheat Engine**. La guía está basada en el tutorial original de **Rydian**, me he basado en sus conocimientos para redactar el siguiente tutorial, os dejo el índice:

- > Introducción
- Buscar Address
 - Conceptos Básicos
 - VEH Debugger
- Pointers

 \geq

- Creando el cheat GODMODE (intento fallido)
- Shared Codes
 - Diseccionar Shared Code
 - Creando el cheat GODMODE (auto-assemble)
- Diseccionar Estructuras
- Localizar estructuras dinámicas
 - AoB to Memory Data
 - Script VBS AoB Pattern generator
- Bonus: Comparar estructuras con WinMerge
- TeleportHack
- Despedida

BUSCAR ADDRESS

CONCEPTOS BÁSICOS

Primero de todo voy a presentar lo que será nuestro objeto de estudio, nada más y nada menos que **Hack, Slash, Loot**. Un juego tipo roguerlike basado en un sistema de combates por turnos que nos ayudará a entender cada una de las partes del tutorial. El juego es un sencillo ejecutable de apenas 10MB que no necesita instalación. No puedo postear el link directo de descarga ya que el juego es de pago (aprox 5€), pero creo que no os costará demasiado hacer una búsqueda en "la baía del barco pirata" para encontrarlo. En el tutorial trabajaremos con la versión **8.0** de **Hack, Slash, Loot**.



Para empezar, yo he seleccionado el arquero, pulsamos el último botón de la derecha para iniciar el juego. Puedes seleccionar otro personaje si lo deseas. El juego se maneja con el ratón; es un roguerlike así que al tratarse de un juego "por turnos" no nos tendremos que preocupar en pausar el juego mientras lo reversamos, **Hack, Slash, Loot** (a partir de ahora **HSL**) es el juego ideal para explicar en un tutorial como éste:



Abajo del todo aparecen los stats de mi jugador, primero de todo hay que marcarse un objetivo, en mi caso nuestro objetivo será crear un cheat de tipo **godmode** (invulnerable). Cada juego es diferente, y cada cheater trabaja de forma diferente. Hay varios caminos que conducen al mismo lugar, cualquier comino es válidos siempre y cuando consigamos cumplir nuestro objetivo. Cuando un juego muestra claramente los puntos de vida (HP) a mi me gusta localizar el address que almacena la vida y bloquearla, otro método válido sería buscar la instrucción que modifica la vida y nopear, otro método es localizar los puntos de vida que te van a restar al recibir un golpe y setearlos a 0. Hay muchos métodos/caminos, y todos son válidos. Yo os explicaré los **3 métodos**.

TRABAJANDO CON VEH DEBUGGER

Empezaremos abriendo nuestro **Cheat Engine 6.3** (a partir de ahora CE), os recomiendo que configuréis las opciones de **CE**, tiene 3 debuggers (2 por software y uno tipo ring0 kernel-mode), a mi me gusta el **VEH**, es practicamente indetectable para los sistemas de anti-cheat, así que nos vamos a "Setting" y modificamos las opciones tal que así:

🖏 Cheat Engine settings	×
General Settings Tools Hotkeys Unrandomizer Scan Settings Plugins Debugger Options Extra	Prefered breakpoint method Hardware Breakpoints (Max 4) Int3 instructions (Execute BP only, falls back to hardware bp) (Unlimited) Page exceptions (Extremely slow to unplayable, buggy, best used if nothing else works. Does not use debug registers) Single line assembler: Replace incomplete opcodes with nops Override existing breakpoints when setting breakpoints Size of stack to record on "Find what routines" 4096 Bytes Debugger method Use windows debugger Use kernelmode debugger (Requires DBVM) Debugger interface config Thread following: Poll for threads Hook thread-create and destroy api's Use processwatcher
	OK Cancel About CE

Una vez tenemos habilitado el VEH Debugger, procedemos a abrir el proceso del HSL



En los stats de nuestro jugador tenemos **30 puntos de vida**, así que realizaremos una búsqueda. No conocemos el funcionamiento del juego, no sabemos de que forma almacena los datos (integer, float, double, ...), mi recomendación y experiencia me dice que probablemente el juego almacene los valores en formato Integer (Long) es decir, **4-bytes**. Procedemos a realizar la primera búsqueda:

First Se	can Next Scan
	Value:
📃 Hex	30
Scan Type	Exact Value 🔹
Value Type	4 Bytes 🔹

En mi caso han aparecido **990 direcciones con el valor 30**. Lo que realizaremos ahora es buscar algún enemigo y dejaremos que nos haga daño para reducir nuestros puntos de vida, por suerte he aparecido al lado de un enemigo, así que haré click con el ratón encima de mi personaje para "pasar el turno" y conseguir que me hagan daño:



Bien, el maldito **kobold** me ha pegado una zurra y ahora mi personaje tiene 14 puntos de vida, así que realizo la segunda búsqueda en CE:

Found: 2				
Address	Value	Previous	New Scan Next Scan	l
0101FD98 03651910	14 14	14 14	Value:	
			Scan Type Exact Value	•
			Value Type 4 Bytes	-

Bien! **De las 990 direcciones encontradas al inicio, ahora solo tengo 2!** No necesito hacer más búsquedas, solo tengo que probar cual de las 2 direcciones almacena la vida. Hacemos click derecho en la primera dirección y selleccionamos la opción "Change value of selected address", en el cuadro emergente ponemos un número cualquiera, por ejemplo 40 y miramos si nuestra vida a aumentado.

Fou	ind: 2						
A	ddress	Va	alue	Previous	New 3	Scan Ne	kt So
	101FD98 3651910		Add selec	ted addresses	to the add	resslist	_
ľ	5051710		Change value of selected addresses				
			Browse this memory region			Ctrl+B	F
			Disassemble this memory region			Ctrl+D	br
		_	Kemove s	elected address		Ctrl+Del	Ē

En mi caso he fallado, la dirección **0101FD98** no es la correcta, ya que si establezco su valor a 40, automáticamente se vuelve a poner con 14. Así que pruebo con la segunda:



Sí! Como podemos observar, **la vida ha aumentado a 40, así que en mi ordenador, la dirección 03651910 almacena los puntos de vida.** El único problema que tenemos ahora es que **esa dirección no es estática**. Si apagamos y encendemos el juego de nuevo, la dirección de los puntos de vida habrá cambiado por otra y tendría que volver a buscarla. ¿Cómo podemo evitar tener que buscar siempre una dirección no estática? Con los **Pointers**. Siguiente capítulo...

POINTERS

No voy a extenderme mucho hablando de los punteros (**pointers**), pero básicamente son una especie de "cajita" en la que el juego almacena la dirección de otro **address**. Nuestro objetivo es localizar el **pointer** (la cajita) que almacena la dirección de nuestra vida, así aunque la dirección cambie, **el pointer siempre apuntará a la dirección correcta.**

Para ello utilizaremos una función propia del **CE** llamada **Pointer Scan**. Necesitamos primero de todo localizar la dirección actual de nuestra vida, en mi ejemplo es la dirección **03651910**. Hacemos doble-click encima de la direcció encontrada para mandarla en la parte inferior, pulsamos botón derecho encima y seleccionamos "Find out what accesses this address":

Active	Description	Address	Туре	Value	
	VIDA	03651910	Delete this r Change reco Browse this	record ord memory region	Ctr
			Show as sig Show as her Change Col	ned kadecimal lor	0
Advand	ced Options		Toggle Sele	cted Records	Sp
			Find out wh	n for this address nat accesses this address nat writes to this address	\bigcirc

Nos preguntará si queremos iniciar el debugger, le decimos que sí, aparecerá una ventanita pequeña. Ahora volvemos al juego y jugamos un poco, movemos el personaje por la pantalla y luchamos con algún enemigo (intentando que nos hagan un poco de daño), tras recibir unos golpes volvemos a la ventanita del **CE** y encontraremos algo similar a esto:

Count	Instruction	
1070	0054A107 - 88 40 50 - mov eax,[eax+50]	
3	00550474 - 29 43 50 - sub [ebx+50],eax	
3	00550477 - 83 7B 50 00 - cmp dword ptr [ebx+50],00	
3	00550A73 - 83 78 50 00 - cmp dword ptr [eax+50],00	
0055046 0055047 0055047 0055047 0055047	E - 88 5D 08 - mov ebx,[ebp+08] 1 - 88 45 0C - mov eax,[ebp+0C] 4 - 29 43 50 - sub [ebx+50],eax << 7 - 83 7B 50 00 - cmp dword ptr [ebx+50],00 B - 7F 0C - jg HackSlashLoot.exe+150489	*
EAX=000 EBX=038 ECX=000 EDX=000 ESI=036 EDI=000 ESP=010 EBP=010 EIP=005	000001 0518C0 000000 00000C 518C0 0018B 01FDC4 01FDC4 01FDCC 50477	

En mi caso el debugger ha encontrado 4 instrucciones que han accedido a la dirección **03651910**. Lo que tenemos que realizar ahora es **buscar un patrón repetitivo**. A simple vista me llama la atención el patrón [???+50]. En cada línea hay un [eax+50] o [ebx+50] ¿Qué significa? Hacemos click en una instrucción, yo he escogido la instrucción **SUB**, abajo aparece una porción de las instrucciones así como el estado de los registros:

Count	Instruction	
1070	0054A107 - 88 40 50 - mov eax,[eax+50]	
3	00550474 - 29 43 50 - sub [ebx+50],eax	
3 3	00550477 - 83 7B 50 00 - cmp dword ptr [ebx+50],00 00550A73 - 83 78 50 00 - cmp dword ptr [eax+50],00	
0055046 0055047 0055047 0055047 0055047	E - 88 5D 08 - mov ebx.[ebp+08] 1 - 88 45 0C - mov eax.[ebp+0C] 4 - 29 43 50 - sub [ebx+50].eax << 7 - 83 78 50 00 - cmp dword ptr [ebx+50].00 B - 7F 0C - jg HackSlashLoot.exe+150489	*
EAX=000 EBX=038 ECX=000 EDX=000 ESI=036 EDI=000 ESP=010 EBP=010 EIP=005	000001 3518C0 000000 518C0 00188 01FDC4 01FDCC 50477	

[ebx+50] se traduce como 036518C0 + 50 (Hexadecimal), así que abrimos nuestra calculadora de Windows (modo Programador) y realizamos la suma:

📑 Cal	culadora						
Ver	Edición	Ayuda					
						3651	1910
000	00 000	30 00e	0000 00	0000	0000	0000	0000
000	00 00:	11 011	0 0101	47 0001	1001	0001	0000
				- 15			
() I	lexa 🐔		Mod A	MC	MR	MS M+	- M-
0	Dec) D		CE	c +	

El resultado de dicha suma es **3651910** que se corresponde a nuestra dirección con los **puntos de vida**. Todo esto lo hemos realizado para conocer que tipo de pointer tendremos que buscar, **os recomiendo que abráis un documento de texto para ir anotando los valores**. En ésta instrucción, **EBX apunta a la estructura base de nuestro jugador**. El **offset +50** de dicha estructura almacena los puntos de vida, tal que así:

EBX = 036518C0 (PLAYER STRUCTURE)

EBX+50 = 3651910 (PLAYER HP VIDA)

El pointer que tenemos que buscar **ha de terminar con el offset +50** ya que es el patrón que hemos visto con el debugger, anotad en vuestro fichero algo así como:



Cerramos el debugger y volvemos a la ventana principal del **CE**, con nuestro address mostrando la vida del jugador. Hacemos click derecho encima del address y seleccionamos **Pointer Scan**:

Active	Description	Address	Туре	Value	
	VIDA	03651910	A Rutas Delete this re	24 ecord	
			Change reco	ord	
			Browse this memory region		
			Show as sign	ned	
			Show as hex	adecimal	
			Change Col	or	
Advanc	ed Options		Set/Change	hotkeys	Ct
Advanc			Toggle Selec	cted Records	S
			Pointer scan	for this address	
			Find out wh	at accesses this address	

Se nos abrirá una ventana con muchas opciones. En la parte superior nos pregunta que dirección queremos buscar. Automáticamente **CE** ha rellenado el address de nuestra vida (**3651910**) ya que lo que queremos es buscar cualquier pointer que apunte a nuestra vida. Más abajo tendremos que cambiar las opciones que he señalado en rojo. Primero le indicaremos al pointer-scan que **nuestro pointer debe terminar con el offset +50** (es lo que descubrimos con el debugger!) Luego parametrizamos la complejidad del pointer, para juegos de éste tipo, con un nivel entre 1-5 es suficiente, el tamaño puede ir desde 1024 a 2048. Para nuestro ejemplo yo he puesto **nivel = 3 y tamaño = 1024**.

[La foto está en la página siguiente]

🐔 Pointerscanner scanoptions 📃 🖃 💌
Address to find: O Value to find:
03651910
Addresses must be 32-bit alligned
Only find paths with a static address
Don't include pointers with read-only nodes
Stop traversing a path when a static has been found
Pointer path may only be inside this region:
From To:
00000000 FFFFFFF
 Improve pointerscan with gathered heap data Only allow static and heap addresses in the path First element of pointerstruct must point to module (e.g vtable) No looping pointers Max different offsets per node: 2 Allow stack addresses of the first thread(s) to be handled as static
Number of threads from oldest to newest: 2 Max stackoffset to be deemed static enough: 4096 Stack addresses as ONLY static address
Use pointermap from previous pointerscan Pointers must end with specific offsets 50 Add Remove
Nr of threads scanning: 3 Normal -
Maximum offset value: 1024 Max level 3
OK Cancel

Pulsamos OK y nos preguntará donde guardar el fichero de pointers, **mi recomendación es que hagáis una sub-carpeta para almacenar el pointer, yo le he puesto el nombre de vida**. Tras unos segundos se iniciará el scaneo de pointers y nos dará un resultado:

🗱 Pointer scan : vida.PTR				
File Pointer scanner				
pointercount:85				
Base Address	Offset 0	Offset 1	Offset 2	Points to:
"HackSlashLoot.exe"	50			03651910
"HackSlashLoot.exe"	3A4	38	50	03651910
"HackSlashLoot.exe"	9C	39C	50	03651910
"HackSlashLoot.exe"	180	14	50	03651910
"HackSlashLoot.exe"	1C	50		03651910
"HackSlashLoot.exe"	3C	9C	50	03651910
"THREADSTACK0"-0	3C	9C	50	03651910
"HackSlashLoot.exe"	3D0	9C	50	03651910
"HackSlachl ont ava"	3EC	9C	50	03651910

Wow 85 punteros. En algunos juegos, la primera búsqueda de pointers puede devolver más de 5 millónes de resultados (es normal), para lograr encontrar el pointer correcto se necesitan realizar varios escaneos consecutivos.

Bien, ya hemos realizado la primera búsqueda y nos ha devuelto **85 punteros**. Lo que realizaremos ahora es **cerrar el juego por completo y lo volveremos a abrir**. Volveremos abrir el proceso de **HSL** y realizaremos de nuevo la búsqueda manual del address con los puntos de vida:

Active	Description	Address	Туре	Value
	VIDA ANTIGUA	03651910	4 Bytes	1
	VIDA	03838880	4 Bytes	23

Como podéis observar, la antigua dirección 03651910 ya no muestra la vida real del jugador, ahora la vida se almacena en la nueva dirección 03838880 (que abremos buscado manualmente tal y como hemos visto en el inicio de éste tutorial). Ya hemos encontrado la nueva dirección con los puntos de vida, así que pulsamos click derecho encima de la dirección y seleccionamos Pointer Scan de nuevo:

Active	Description	Address	Туре	Value	
	VIDA ANTIGUA	03651910	4 Bytes	1	
	VIDA	03838880	Delete this r Change reco Browse this Show as sign	ecord ord memory region ned	Ct
Advan	ced Options	_	Show as hex Change Col Set/Change Toggle Sele	adecimal or hotkeys cted Records	Ct
			Pointer scan Find out wh	for this address at accesses this address	

ATENCIÓN! Ahora se ha vuelto a abrir la misma ventana de antes, con las opciones del pointer. Lo que hay que hacer es CERRAR ésta ventana ya que no queremos iniciar una nueva búsqueda, si no continuar la búsqueda con los resultados anteriores:

	🗱 Pointer scan				
	File Pointer scanner				
	New Ctrl+N				
	Open Ctrl+O				
🐔 Oper	existing file				
00	🗢 퉬 🕨 Tutorial 🕨 pointers				
Organ	izar 🗙 Nueva carpeta				
orgu					
	Nombre				
1	vida.PTR				
	10/100-				
	1004000				

Se nos habrán cargado los pointers anteriores (en mi caso 85), así que realizaremos una búsqueda utilizando estos 85 punteros de referencia:

	🐔 Pointer scan : vida.P	TR				
	File Pointer scanner					
Scan for pointer	Ctrl+P	•		pointe	rcount:85	
Rescan memory - Removes pointers not pointing to the right	address Ctrl+R		Offset 0	Offset 1	Offset 2	Points to:
	"HackSlashLoot.exe	9"	50			038388B0
	HackSlachl ont ovo	. "	344	38	50	0000050

Se nos abrirá una ventana más pequeña, aquí **indicaremos el valor de la nueva dirección que debería tener el pointer**, introduciremos la nueva address que hace referencia a los puntos de vida, en mi caso **038388B0**:

Rescan pointerlist		
Address to find: O Value to find:		
03838880		
Only filter out invalid pointers		
Delay rescan for 0 seconds		
Repeat rescan until stopped		
Lua filter. function RescanFilter (base, offsets, targe		
🔲 Base pointer must be in range		
000000000000000 and FFFFFFFFFFFFFFF		
Must start with offsets		
Must end with offsets		
OK Cancel		

Pulsamos **OK** (nos preguntará donde guardar los resultados, podéis guardar con el mismo nombre o con un nombre nuevo) tras unos segundos nos reducirá los resultados según el valor buscado:

🐔 Pointer scan : vida.PTR				
File Pointer scanner				
4 Byte 🔹	ро	intercount:1		
Base Address	Offset 0	Offset 1	Offset 2	Points to:
"HackSlashLoot.exe"+007B368	3 50			038388B0 = 23

Bieeeen, **1 solo resuntado**. Seguro que ese es nuestro **pointer**! Además **el offset termina en +50** tal y como hemos configurado. Hacemos doble-click en el resultado y el pointer se mandará a nuestra tabla de **CE**:

Active	Description	Address	Туре	Value
	VIDA ANTIGUA	03651910	4 Bytes	1
	VIDA	038388B0	4 Bytes	23
	pointerscan result	P->038388B0	4 Bytes	23

Podemos ver como **CE** muestra los pointers con el carácter **P->**, si ahora reiniciamos el juego, el pointer nos mostrará la dirección de nuestra vida, podremos editar, congelar o trabajar con la dirección.

Hasta aquí el tutorial sobre **pointers**, practicad y veréis que no es tan complicado como parece.

(PRIMER INTENTO FALLIDO)

Bueno, ya que hemos encontrado el address de nuestra vida y que, además, **tenemos el pointer estático** de dicho address, podemos establecer un valor y marcar la casilla para "congelarlo", así obtendríamos lo que sería una especia de godmode:



Lo único que no me gusta de éste método es que si luchamos contra un enemigo muy poderoso y de un solo golpe nos quita -**700 puntos de vida... terminaremos muriendo** ya que la velocidad de refresco del **CE** nunca será superior a la del juego. Para evitar eso y conseguir un **godmode** más real podemos hacer muchas cosas, pero yo recomiendo las siguientes:

- Detectar el valor de daño que nos aplicarán y establecerlo siempre a 0
- Detectar la instrucción que modifica o resta los puntos de vida y cambiarla por un NOP

Ambos métodos son correctos y yo los aconsejo ya que son "mejores" que setear un **pointer** a un valor fijo. Para ello pulsaremos click derecho en nuestro **pointer** (que apunta a la address de los puntos de vida) y miraremos que instrucciones escriben en dicha dirección:

Active	Description	Address	Туре	Value	
	pointerscan result	P->037490B0	Delete this Change re	s record cord	[
			Browse th	is memory region	Ctrl
			Show as si	igned	
			Show as h	exadecimal	
			Change C	olor	
			Set/Chang	ge hotkeys	Ctrl+
Advanc	ced Options		Toggle Se	lected Records	Spa
			Pointer sc	an for this address	
			Find out w	what accesses this address	
			Find out w	what writes to this address	
					N

Al tratarse de un pointer, **CE** nos preguntará si queremos mirar la instrucción que modifica el pointer o la instrucción que modifica la dirección a la que apunta nuestro pointer. **Siempre, siempre, siempre escogeremos la segunda opción**:

🐔 Cheat Engine Pointer	×
This is a pointer.	
Find out what writes this pointer	
Find what writes the address pointed at by this pointer	

Nos volverá a salir la pequeña ventanita del **debugger**, que estará vigilando a ver que instrucción escribirá sobre el address del pointer (los puntos de vida). Volvemos al juego y jugamos unos cuantos turnos hasta recibir un poco de daño, cuando hayamos recibido daño, volvemos a la ventanita del **debugger** y encontraremos lo siguiente:

🗱 The following opcodes write to 037490B0	
Count Instruction	
1 00550474 - 29 43 50 - sub [ebx+50],eax	
0055046E - 8B 5D 08 - mov ebx,[ebp+08] 00550471 - 8B 45 0C - mov eax,[ebp+0C]	^
00550474 - 29 43 50 - sub [ebx+50],eax << 00550477 - 83 78 50 00 - cmp dword ptr [ebx+50].00	
0055047B - 7F 0C - jg HackSlashLoot.exe+150489	
EAX=00000007	
EBX=03749060 ECX=00000000	
EDX=0000000C	
EDI=0285D990	
ESP=0101FE94 EBP=0101FE9C	
EIP=00550477	

La ventana nos muestra que la instrucción **00550474** ha modificado nuestra vida, ademas nos enseña el código **ASM** que realiza la acción de modificar **sub [ebx+50],eax** y el estado de todos los registros, vámos a interpretar:

POINTER VIDA = 37490B0

INSTRUCCIÓN = SUB [EBX+50],EAX

EBX = 03749060

```
EAX = 7
```

Lo que está ocurriendo aquí es la llamada a la instrucción SUB (substract / restar) el valor de EAX a la dirección [EBX+50]

EBX+50 = 03749060 + 50 = <u>37490B0</u> (Pointer Address Vida)

En resumen, resta el valor de EAX=7 a nuestra dirección de memoria que almacena los puntos de vida, por lo que dicha instrucción me quita 7 puntos de vida.

Perfecto, ahora ya entra la genialidad de cada uno para reversar dicha instrucción. Hay varias formas y todas son válidas:

- Cambiar el SUB por un ADD (así en cada golpe, la vida aumentará)
- Cambiar EAX por 0, quedaría así: sub [ebx+50],0 (así en cada golpe, la vida disminuye en 0)
- Nopear la instrucción cambiando el sub [ebx+50],eax por varios NOP

De las 3 opciones que os he propuesto, la primera de todas parece la más sencilla. Además a parte de conseguir un **godmode**, nuestra vida aumentaría en cada golpe. Vámos a probar. Pulsamos el botón "Show disassembler" para abrir la ventana de **Memory View**:

File Search View Debug Tools Kernel tools					
HackSlashLoot.exe+150474					
Address Bytes Opcode Comment	•				
HackSlashLoot.exe+150474 29 43 50 sub (ebx+50),eax					
HackSlashLoot.exe+150477 83 7B 50 00 cmp dword ptr [ebx+50],00 0					
HackSlashLoot.exe+15047B 7F 0C -ig HackSlashLoot.exe+150489					
HackSlashLoot.exe+15047D 53 push ebx					
HackSlashLoot.exe+15047E 8B 03 mov eax.[ebx]					
HackSlashLoot.exe+150480 FF 90 80010000 call dword ptr [eax+00000180]					
HackSlashLoot.exe+150486 83 C4 04 add esp.04 4					
HackSlashLoot.exe+150489 A1 8C36B800 ▶mov eax.[HackSlashLoot.exe+7B368C]					
HackSlashLoot.exe+15048E FF 731C push [ebx+10]					
HackSlashLoot.exe+150491 FF 7318 push [ebx+18]					
HackSlashLoot.exe+150494 50 push eax					
HackSlashLoot.exe+150495 88.00 mov eak,[eak]					
HackSlashLood.exe+150497 FF 50487 Call dword.ptr [eax+48]					
HackStashLood.exeH3043A 83 L4 UL and esp.UL 12					
HackStashLoot.exe+150450 63 F6 00 Cmp eax,00 0 0					
subtract					
Protect:Read/Write Base=00601000 Size=1000 Module=HackSlashLoot.exe					
address 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF					
00601000 00 08 00 00 60 B2 BC 00 00 00 00 01 00 00 00`					
00601040 00 00 00 00 A5 EC BC 00 A5 EC BC 00 01 00 00 00					
00601050 06 00 00 00 00 00 00 00 00 00 00 00 00					
00601060 00 00 00 00 00 00 00 00 01 00 00 00 00					
006010C0 00 00 00 00 00 00 00 00 01 00 00 00 00					
006010D0 00 00 00 00 00 00 00 00 00 00 00 00					
UUGUIUEU UU UU UU UU 11 00 00 00 AE EC BC 00 00 00 00 00					
006010F0 00 00 00 00 00 00 00 00 00 00 00 00					
	-				

CE ya nos ha posicionado en la instrucción **00550474**, así que hacemos click derecho en la instrucción y seleccionamos "Assemble" para editarla:

HackSla	ashLoot.exe+150474	•
Opcod	e	Commen
sub	[ebx+50].eax	6 · · · · ·
cmp	dword ptr [ebx+5	Go to address
-19 Dueb	HackblashLoot.	Replace with code that d
mov	eax.[ebx]	Addes the sede list
call	dword ptr [eax+0	Add to the code list
add	esp.04	Assemble
mov	eax,[HackSlash]	Copy to clipboard
nush	[eby+1C]	

Y cambiamos la palabra **SUB** por **ADD**:

Cheat Engine single-line assembler				
Type your assembler code here: (address=00550474)				
add [ebx+50],eax				
ОК	Cancel			

Os recomiendo que hagáis click derecho de nuevo sobre la instrucción y os guardéis la dirección en el fichero de texto:

	Address	Bytes	Opcode
	HackSlashLoot.exe+15047 HackSlashLoot.exe+15047 HackSlashLoot.exe+15047 HackSlashLoot.exe+15047 HackSlashLoot.exe+15047 HackSlashLoot.exe+15048 HackSlashLoot.exe+15048	4 01 / 2 50 7 Go to address 7 B 7 Replace with 7 Add to the co 9 Assemble	add tel s code that does notl ode list
Bytes+Opco	des	Copy to clipb	oard
Bytes Opcodes Bytes only (n	io address) Ctrl+Alt+C	Change regis Toggle break Break and tra	ter at this location point ce instructions
J I I	info.txt - NFOPad* Archivo Editar Configurac DIRECCION RESTAR VID HackSlashLoo PLAYER STRUCTURE +50 =	iones Ver Ayuda A t.exe+150474 HP Vida	

Así **tendréis guardada la dirección para más tarde y ahorraremos tiempo**, en mi caso es **HackSlashLoot.exe+150474**. Cerramos el **Memory View** y la ventanita del **debugger**, dejando solo la ventana principal del **CE** y el juego. Iniciamos un nuevo combate para ver si funciona nuestro "hack":



670 puntos de vida! En lugar de disminuir ha aumentado! Bien somos unos **hackers-cheaters-crackers** de la elite profesional, pero... si intentas matar a tu enemigo haciendo click encima del **kobold**, verás que por una extraña circunstancia no puedes matarlo... ¿Qué raro, no? Si miramos bien, el juego nos muestra la vida de nuestro enemigo si lo seleccionamos con el ratón:

🔊 Kobo I	d
Sn i ve	Hing
v 15∕1	Únarmed
⊽ 0	🛡 76

Horror! La vida del enemigo también aumenta! ¿Cómo es eso posible? Bueno, esto por desgracia suele ocurrir. El programador del juego está utilizando la misma función para restar vida al jugador y a los enemigos, por eso al haber cambiado el SUB por el ADD, tanto la vida del jugador como la de los enemigos aumenta en cada golpe. Éste efecto se le conoce como Shared Code (código compartido) ya que la instrucción HackSlashLoot.exe+150474 es compartida por más de una dirección de memoria. ¿Hay alguna forma de solucionar éste problema? Sí, sigue leyendo...

SHARED CODES

Primero de todo vamos a dejar el juego como estaba, quitando el ADD que pusimos anteriormente por el SUB original, abrimos el Memory View:

				Start	000
				Stop	7ff
				📝 Writ	able
				📃 Сор	yOnWrite
				📝 Fasl	t Scan 4
			\mathbf{X}	📃 Pau	ise the gan
Memo	ory View			0	
Active	Description	Address	Ту	ре	Value
	pointerscan result	P->037490B0	4 E	lytes	670

Si hemos cerrado el **CE** quizás ya no estemos encima de la instrucción original, copiamos la instrucción de nuestras notas (en mi caso HackSlashLoot.exe+150474) y hacemos:

		HackSlashLoot.e	xe+1C			
Address	Bytes	Opcode				
HackSlashLoot.exe+1	00.00	le lucel bbc				
HackSlashLoot.exe+1	Go to address	N				
HackSlashLoot.exe+2 HackSlashLoot.exe+2	Back	6				
HackSlashLoot.exe+2 HackSlashLoot.exe+2	Replace with code that does nothing					
HackSlashLoot.exe+2	Add to the code lis	st				
Goto	Address	×				
	Fill in the address y	ou want to go to				
Hack	SlashLoot.exe+150474	•				
	ОК	Cancel				

Nos situaremos delante de la instrucción, hacemos **doble-click encima** (o click derecho / "Assemble") y cambiamos el **ADD** por el **SUB** original, así todo estará en su sitio.

Ahora lo que os voy a enseñar es un método para **comprobar si una misma instrucción modifica más de una dirección de memoria**, para ello pulsamos botón derecho encima de la instrucción y seleccionamos "Find out what addresses this instruction accesses"

HackSla	shLoot.exe+150474	
Opcode	2	Comment
sub cmp -ig push mov call add mov push push push mov call add cmp	ebx+50,eax dword ptr [ebx+50],00 HackSlashLoot.exe+1504 ebx eax,[ebx] dword ptr [eax+000001B0 esp,04 eax,[HackSlashLoot.exe+ [ebx+10] [ebx+18] eax eax,[eax] dword ptr [eax+48] esp,00 eax,00	Go to address Back Replace with code that does nothing Add to the code list Assemble Copy to clipboard Change register at this location Toggle breakpoint Break and trace instructions
je	HackSlashLoot.exe+1504	Find out what addresses this instruction accesses

HackSla	shLoot.exe+150474		
Opcode	2	Comment	A
sub	(ebx+30),eax		
cmp -ig push mov	dword ptr [ebx+50].00 HackSlashLoot.exe+150489 ebx eax,[ebx]	Changed Addresses The following addresses selected	s have been changed by the code you
call add push push call add cmp je	dword ptr [eax+00000180] esp,04 eax,[HackSlashLoot.exe+78368 [ebx+1C] [ebx+18] eax eax,[eax] dword ptr [eax+48] esp,0C eax,00 HackSlashLoot.exe+1504E4	Address Va	alue Count
size=10 09 0A 00 00 00 00 EC BC 00 00	Subtract 00 Module=HackSlashI 0B 0C 0D 0E 0F 0123 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Stop	4 Bytes 🔹

Si lo hemos hecho bien, la instrucción quedará marcada en **verde** y aparecerá una ventana. Sin cerrar la ventana, volvemos al juego e intentamos forzar que dicha instrucción trabaje. Para ello iniciaos un combate hasta **recibir algo de daño y causar nosotros daño al enemigo:**

tacks Kobold.	Address	Value	Count
kes 6 damage.	104D3490	70	1
b <mark>acks Anchen.</mark> Kes 1 damage.	03749080	663	1
ips go. 🚬 📕 🎘 🚛			
Lacks Arche <mark>ntfick <mark>Mobolo</mark> Sses Archer.</mark>			
ips 90.			
tacks Anchen.	Stop	4 Bytes	•
Attack Kobold	· · · · · · · · · · · · · · · · · · ·		
nd elf Kobold	· · · · · · · · · · · · · · · · · · ·		
Attack			
		← D que pusimos	

Y aquí tenemos el resultado, han aparecido 2 direcciones que han sido accedidas por la misma instrucción, la primera (104D3490) tiene valor 70 y se corresponde con la vida del enemigo (ver foto). La segunda dirección (037490B0) es la dirección de mi vida, además su valor coincide con la vida de mi personaje (ver foto). Está más que claro que ésta instrucción ha modificado ambas direcciones, por lo que si cambiamos el SUB por un ADD afectará tanto a mi vida como a la vida de los enemigos.

Bueno, pulsamos **STOP** para que **CE** deje de observar la instrucción (la instrucción ya no estará marcada en verde) y dejamos la ventanita con las 2 direcciones abierta). Ahora que ya conocemos el método para saber cuando estamos delante de un **Shared Code** os explicaré como **diseccionar la estructura del jugador** y la del enemigo para buscar diferencias y poder aplicar un **ADD** en el caso que sea el jugador y un **SUB** en el caso de un enemigo.

DISECCIONAR SHARED CODE

Seguimos! Hemos encontrado una instrucción del tipo Share Code", os dejo el resumen:

DIRECCIÓN	=	HACKSLASHLOOT.EXE+150474
OPCODES	=	SUB [EBX+50],EAX
ΤΙΡΟ	=	SHARED CODE

Sabemos que el **offset +50** de la estructura es donde el juego almacena la vida, dependiendo del valor de **EBX** la instrucció modificará la dirección del jugador o la del enemigo. Está clarisimo que tendremos que **ver el estado de los registros** para saber la dirección afectada, para ello hacemos click drecho en nuestra dirección del jugador y seleccionamos "Show register states":



Se nos abrirá una mini ventana con el estado de los registros, hacemos lo mismo con la dirección del enemigo y ponemos las 2 ventanas una al lado de la otra:



Nos fijamos en los registros **EBX**, naturalmente EBX apunta al inicio de la estructura de cada jugador (nuestro player y el enemigo). Si miramos con más atención, el registro **ESI** tiene el mismo valor que **EBX** en el caso del Player pero en el caso del Enemigo **ESI** no vale lo mismo que **EBX**.

Pues de ésta forma tan sencilla acabamos de inventarnos un método para conocer cuando la instrucción está modificando el address del player o del enemigo, dicho método es una simple comparación:

CMP EBX,ESI

JNE ENEMIGO

O lo que es lo mismo, **comparamos el valor de EBX con ESI y saltaremos a "ENEMIGO" si no son iguales.** ¿Qué os parece? Habría otras formas para diferenciar ambas estructuras, pero la que a mi se me ha ocurrido es ésta. Ahora solo falta decirle a **CE** que sepa diferenciar el **Shared Code** y nos haga un **ADD** o un **SUB** cuando nosotros queramos.

(SEGUNDO INTENTO) : AUTO-ASSEMBLE

Cerramos la ventanita para quedarnos delante del **Memory View** con la instrucción seleccionada. Vamos al menú "Tools / Auto-Assemble":

	🐔 Memory Viewer													
	File Search View	Debug Tools) Keri	nel to	ools									
Allocate	e Memory	Ctrl+Alt+M				Hac	cSlas	hLoo	ot.exe	2+15(0474			
Scan fo	r code caves	Ctrl+Alt+C				Орс	ode							
Fill Mer	morv	Ctrl+Alt+F	20			sub		ebx	+50),e	вах	5010	0		
Create	Thread	Ctrl+ Alt+ T	0			стр іл		Hack	o ptr Slas	leox+ hl oo	Fouju Lexe	JU +150	489	
create	meau	CultAitt				pusl	n	ebx						
Dissect	code	Ctrl+J				moy		eax.	ebx	,				
Dissect	data/structures	Ctrl+D	10000	J		call add		dwor	d ptr 14	eax	-0000	JUTRI	IJ	
Dissect	window(s)	Ctrl+W	300		- i	moy		eax.	Hack	Slas	hLool	t.exe-	+783680]
Dissect	PE headers	Ctrl+H				pusl	n	[ebx	+10]					
						pusi	า า	lebx-	+18]					
Pointer	scan	Ctrl+P				MOY	•	eax.	eax					
Structu	re spider	Ctrl+Alt+P				call		dwor	d ptr	eax	48]			
Ultimap	þ	Ctrl+Alt+U				add		esp. eax	JC 10					
Watch i	memory allocations	Shift+Ctrl+M				je		Hac	(Slas	hLoo	t.exe-	+150	4E 4	
Find sta	atic addresses	Ctrl+Alt+S					s	ubt	rac	t				
Lua Eng	gine	Ctrl+L	04D3 96	3000 97) Si 98	ize: 99	=AE(9A	000 9B	9C	9D	9E	9F	0123	4563
Inject D	DLL	Ctrl+I	00 4D	00	00 30	00	00 4D	00	00 A0	00 2C	00 61	00	F	5M
Auto As	ssemble	Ctrl+A	61	00	60	0D	61	00	00	00	00	00	L.	`.a
	10403400 00 00		00	00	85	43	00 0B	BD	00	00	00	00	,а.	,а

Se nos abrirá un editor de **Auto-Assemble** (a partir de ahora **AA**), podemos escribir a mano todo el script pero **CE** tiene un par de plantillas que nos ahorrarán mucho trabajo, hay que usar los siguientes menús, hay que hacerlo en el mismo orden que os explicaré:

- 1. "Template / Cheat Table framework code"
- 2. "Template / Code Injection"
- 3. Pulsar OK para aceptar la dirección de nuestra instrucción

Si lo hemos hecho bien obtendremos el siguiente código automático:

[foto en la siguiente página]

🐔 А	uto assemble	×
File	View Template	
1	[ENABLE]	
2	//code from here to '[DISABLE]' will be used to enable the cheat	
3	alloc(newmem, 2048)	
4	label (returnhere)	
5	label(originalcode)	
6	label(exit)	
7		
8	newmem: //this is allocated memory, you have read, write, execute access	
9	//place your code here	
10		
11	originalcode:	
12	sub [ebx+50], eax	
14	amp dword per [ebx+30],00	
15	evit.	
16	jmp returnhere	
17		
18	"HackSlashLoot.exe"+150474:	
19	jmp newmem	
20	nop	
21	nop	
22	returnhere:	
23		
24		
25		
26		
27	[DISABLE]	
28	//code from here till the end of the code will be used to disable the cheat	
29	dealloc(newmem)	
30	"HackSlashLoot.exe"+1504/4:	
31	stub [ebx+50], eax	
33	(/2) + , db 20 42 E0 82 78 E0 00	
35	//AIL: ab 29 43 50 63 /b 50 00	-
		•
	Execute	

Ahora vamos al menú "File / Assign to current cheat table" y luego "File / Exit". Si lo hemos hecho bien, la ventana del **Auto-**Assemble se habrá cerrado y tendremos un script en la ventana principal del **CE**:

			🔪 📄 Pau	use the game while :		
Memo	ory View		\bigcirc			
Active	Description	Address	Туре	Value		
	pointerscan result	P->037490B0	4 Bytes	669		
	Auto Assemble script			<script></script>		

Hacemos **doble click encima del script** y se volverá a abrir la ventana del **Auto-Assemble**, desde aquí ya podremos modificar el script para conseguir un **godmode** como dios manda (valga la redundancia), vámos a identificar cada parte del código:

El script está separado por 2 secciones grandes llamadas [ENABLE] y [DISABLE]. El código de la parte [ENABLE] se ejecutará cuando el script esté activo. Cuando desactivemos el script, se ejecutarán las instrucciones de la sección [DISABLE].

En la parte de **[ENABLE]** empieza con un **alloc()** que sirve para reservar una sección de memoria. Por defecto **CE** nos asignará **2kbytes para inyectar código ASM**, dicha sección de código estará bajo la etiqueta **NEWMEM**. Luego encontramos **3 etiquetas** que sirven para identificar partes del código:

- Returnhere = Indica el final del código
- Originalcode = Indica la parte original del código, en nuestro caso el SUB
- Exit = No se utiliza, es lo mismo que Returnhere

Teniendo éstas 3 partes bien identificadas queda muy claro que escribiremos nuestro código bajo la etiqueta **NEWMEM**, el resto no lo queremos modificar. En el espacio que tenemos entre **NEWMEM** y **ORIGINALCODE** escribimos lo siguiente:

CMP EBX,ESI

JNE ORIGINALCODE

MOV EAX,0

¿Que significan éstas líneas? Básicamente lo que estamos realizando primero es **comparar el registro EBX con ESI**. Anteriormente dijimo que **si EBX = ESI significa que es el jugado, en cambio si EBX != ESI entonces es el enemigo**. Luego lanzamos un salto condicional **JNE** (Jump if Not Equal) es decir "salta si no es igual", si **EBX** no es igual a **ESI** entonces saltará a **ORIGINALCODE** y ejecutara el **SUB**, en cambio si **EBX** es igual a **ESI** (jugador) no saltará y se ejecutará la instrucción **MOV EAX,0** es decir, **EAX=0**. Con esto lo que conseguimos es que cuando se ejecute el **SUB**, el registro **EAX valdrá 0** y no nos restará la vida. El **script** te ha de quedar así:

[foto en la siguiente página]

Ę A	uto Assemble edit: Auto Assemble script	
File	View Template	
1	[ENABLE]	
2	//code from here to '[DISABLE]' will be used to enable the cheat	
3	alloc(newmem, 2048)	
4	label (returnhere)	
5	label(originalcode)	
6	label(exit)	
7		
8	newmem: //this is allocated memory, you have read, write, execute access	
9	//place your code here	
10	cmp ebx,esi	
11	jne originalcode	
12	mov eax,0	
13		
14	originalcode:	
15	<pre>sub [ebx+50], eax</pre>	
16	cmp dword ptr [ebx+50],00	=
17		-
18	exit:	
19	jmp returnhere	
20		
21	"HackSlashLoot.exe"+150474:	
22	jmp newmem	
23	nop	
24	nop	
25	returnhere:	
26		
27		
28		
29	(DTC3DIE)	
21	[DISADLE]	
32	//code from here till the end of the code will be used to disable the cheat	
32	"HackSlashLoot eve"+150474.	
34	sub [eby+50] eav	-
4	Sab [coxios]/cax	•
_		
	ОК	

Ahora activamos el **script** haciendo click en el recuadro y **luchamos con algún enemigo.** Verás que si eres golpeado, tu vida no decrece. En cambio si golpeas a un enemigo, su vida decrecerá con normalidad. **Desactiva el script y abre el Memory View**, hacemos un **GoTo Address "HackSlashLoot.exe"+150474**, aparecerá el código original:

		н	ackSlashLoot.exe+150474
Address	Bytes	Opcode	e
HackSlashLoot.exe+150474	29 43 50	sub	[ebx+50],eax
HackSlashLoot.exe+150477	83 7B 50 00	cmp	dword ptr [ebx+50],00

Ahora pulsamos el script para activarlo y miramos que ocurre en dicha instrucción:

		HackSlashLoot.exe+150474
Address	Bytes	Opcode
HackSlashLoot.exe+150474	E9 87FB6F02	jmp 02C50000
HackSlashLoot.exe+150479	90	nop

El código original ha sido sustituido por un jmp 02C50000 (en tu caso podrá ser otro address), hagamos click derecho en el jmp y seleccionamos Follow para ver que hay ahí:

	02C50000		
Address	Bytes	Opcod	le
02C50000	39 F3	стр	ebx esi
02C50002	0F85 05000000	-jne	02C5000D
02C50008	B8 0000000	mov	eax,00000000
02C5000D	29 43 50	sub	[ebx+50],eax
02C50010	83 7B 50 00	cmp	dword ptr [ebx+50],00
02C50014	E9 620490FD	jmp	HackSlashLoot.exe+15047B

Lo que encontramos tras ese **jmp** es el código **ASM** que hemos inyectado con el **Auto-Assemble script**, aquí se ve muy claro como trabaja **CE**, si desactivamos el script, dicha zona de memoria será borrada.

RECORRER LOS OFFSETS "A MANO"

Ahora ya tenemos un **script** en **Auto-Assemble** que nos permite activar un cheat tipo **godmode** y que además es capaz de diferenciar si se trata de un jugador o de un enemigo. Lo que voy a explicar a continuación es **como diseccionar una estructura para encontrar otros valores interesantes.**

Una estructura en programación consiste en declarar una serie de variables comunes y asignarlas a un "nombre", ejemplo:

STRUCT PLAYER
{
INT ID;
CHAR NAME[10];
FLOAT MANA;
DOUBLE HEALTH;
};

En éste caso he creado una estructura llamada player que contiene **4 variables** (id, nombre, mana, health). Así es como están programados la gran mayoría de video-juegos. Las estructuras están cargadas en memoria (tiempo de ejecución). Nuestro objetivo será conocer el address de la estructura de nuestro jugador para poder diseccionarla con una herramienta (base-address). Sigamos con el tutorial...

Primero de todo trabajaremos con el juego "original", así que el **script de godmode lo dejamos desactivado.** Tenemos ya nuestro **pointer** que apunta a la dirección de los puntos de vida del jugador. Hacemos click derecho encima del **pointer** y seleccionamos "Find out what writes to this address":

Active	Description	Address	Туре	Value	
	pointerscan result	P->03918880	A Dutas	660	
	Auto Assemble script		Delete th	is record	
			Change	record	
			Browse t	his memory region	Ct
			Show as	signed	
			Show as	hexadecimal	
			Change	Color	
			Set/Char	nge hotkeys	Ctr
			Toggle S	elected Records	Sp
			Pointer s	can for this address	
Advanc	ced Options		Find out	what accesses this address	
			Find out	what writes to this address	
					15

Al tratarse de un pointer nos preguntará si queremos mirar la instrucción que escribe encima del pointer o la instrucción que escribe el address de nuestro pointer. **Siempre escogeremos la segunda opción.** Se abrirá la ventana del **debugger** así que volvemos al juego y **dejamos que un enemigo nos golpee**, acto seguido volvemos a la ventanita y nos aparecerá nuestra instrucción famosa:

🗱 The following opcodes write to 039188B0	—
Count Instruction 1 00550474 - 29.43.50 - sub [ebx+50],eax	Replace
	Show disassembler
	Add to the codelist
0055046E - 88 5D 08 - mov ebx,[ebp+08] 00550471 - 88 45 0C - mov eax,[ebp+0C] 00550474 - 29 43 50 - sub [ebx+50],eax << 00550477 - 83 78 50 00 - cmp dword ptr [ebx+50],00 00550478 - 7F 0C - jg HackSlashLoot.exe+150489 EAX=00000001 EBX=03918860 ECX=00000000 EDX=0000000C ESI=03918860 EDI=0019D990 ESP=0101FE94 EBP=0101FE94 EBP=0101FE9C EIP=00550477	More information subtract
	Stop

Si hacemos click en ella nos aparece abajo toda la información. Pulsamos el botón "More Information"

🐔 Extra info 📃 📼 💌	
0055046E - mov ebx,[ebp+08]	
00550471 - mov eax,[ebp+0C]	
>>00550474 - sub [ebx+50],eax	
00550477 - cmp dword ptr [ebx+50],00	
0055047B - jg HackSlashLoot.exe+150489 🦯	
subtract The value of the pointer needed to find this address is probably 03918860	
EAX=00000001 EDX=0000000C EBP=0101FE9C F	ł.
EBX=03918860 ESI=03918860 ESP=0101FE94	1
ECX=00000000 EDI=0019D990 EIP=00550477	J
The registers shown here are AFTER the instruction has been executed OK	

Nos aparece un recuadro con información adicional, he subrallado una frase importante que nos indica **CE**... nos está calculando cual es **la direcció probable de nuestra estructura**, en mi caso nos indica **03918860**, así que hacemos click derecho encima para copiar la dirección:

subtract			
The value of the pe	intor	needed to find this address is nr	pbably 03918860
	hr	Copy easy guess to clipboard	

Perfecto, ya tenemos la **base-address** de la estructura de nuestro jugador, ya podemos diseccionarla! Cerramos la ventana de "More Information" y la ventanita del **debugger**. Nos situamos en la ventana principal de **CE**, pulsamos en **Memory View** y luego en el menú: "Tools / Dissect data & structures":

ŧ	Memory Viewer			
Fi	le Search View Debug	Tools Kernel too	ls	
	Allocate Memory	Ctrl+Alt+M	H	ackSlashLoot.exe+15046D
	Scan for code caves	Ctrl+Alt+C	Opcod	e
	Fill Memory	Ctrl+Alt+F	push mov	ebx ebx,[ebp+08]
	Create Thread	Ctrl+Alt+T	moy	eax,[ebp+0C]
			sub	[ebx+50].eax
	Dissect code	Ctrl+J	cmp	dword ptr [ebx+50],00
	Dissect data/structures	Ctrl+D	-19 push	HackSlashLoot.exe+150489 ebx
	Dissect window(s)	Ctrl+W	mov	eax,[ebx]

Se abrirá una ventana grande de color blanco, en la parte superior podremos escribir una address. Lo que hay que hacer es escribir la **base-address** de nuestra estructura (en mi caso **03918860**):

🖏 Structure dissect	
File View Structures	Structure Options
Group 1	
03918860	
Offset-description	Address: Value

Vámos al menú: "Structures / Define New structure". Nos preguntará que nombre queremos asignar, puedes poner lo que quieras, yo puse **PLAYER**. A continuación nos preguntará si queremos que **CE** rellene automáticamente la estructura:



Le diremos **YES** para ahorrarnos trabajo, jeje. Luego nos preguntará el tamaño. Por defecto **CE** asigna **4096 offsets**. Bajo mi punto de vista son demasiados, así que yo lo bajo a **1024 o 2048**:

Structure define	
Please give a starting size of the struct (You can c	hange this later if needed)
1024	
G	<u>O</u> K Cancel

Finalmente, tras definir el tamaño, **CE** cogerá la **base-address** de la estructura y la **diseccionará 1024 offsets**, obtendremos lo siguiente:

all a second second	-			
🐛 Structure dissect:PLAYER				
File View Structures	Structure Options			
Group 1				
03918860				
Offset-description	Address: Value			
PLAYER				
⊳0000 - Pointer	3918860 : P->00BB1218			
-0004 - 4 Bytes	3918864 : 2			
⊳0008 - Pointer	3918868 : P->0390C580			
⊳000C - Pointer	391886C : P->039128F0			
-0010 - 4 Bytes	3918870 : 3			
-0014 - 4 Bytes	3918874 : 3			
-0018 - 4 Bytes	3918878 : 16			
-001C - 4 Bytes	391887C : 17			
-0020 - 4 Bytes	3918880 : 2			
⊳0024 - Pointer	3918884 : P->03919DB0			
-0028 - 4 Bytes	3918888 : 0			
-002C - 4 Bytes	391888C : 4294967295			
-0030 - 4 Bytes	3918890 : 649			
-0034 - 4 Bytes	3918894 : 40			
-0038 - 4 Bytes	3918898 : 60			
-003C - 4 Bytes	391889C : 50			
-0040 - 4 Bytes	39188AO : O			
-0044 - 4 Bytes	39188A4 : O			
-0048 - 4 Bytes	39188A8 : O			
-004C - 4 Bytes	39188AC : 0			
-0050 - 4 Bytes	39188BO : 668			
-0054 - 4 Bytes	39188B4 : 30			
-0058 - 4 Bytes	39188B8 : O			
-005C - 4 Bytes	39188BC : 0			
-0060 - 4 Bytes	39188C0 : 1			
⊳0064 - Pointer	39188C4 : P->039187E0			
⊳0068 - Pointer	39188C8 : P->03918990			

No nos asustemos, vámos a interpretar la información que nos proporciona **CE**... ¿recuerdas que offset corresponde a los puntos de vida? Puedes mirar las notas o el pointer que tenemos en la tabla:

🔛 info.txt	- NFOPa	d				x		
Archivo	Editar	Configuracione	s Ver	Ayu	da			
DIRECCI	ON RES	STAR VIDA				*		
HackSlashLoot.exe+150474								
PLAYER	STRUCI	TURE			_			
	+50	=	HP V	'ida				

Pues sí, la vida está situata en el offset +50, si miramos la estructura diseccionada veremos que se corresponde al valor del pointer y a los puntos de vida:

🐔 Structure	e dissect:PLAY	/ER		
File View	Structures	Structure Options		
Group 1				
03918860				
011 1				
Offset-descr	iption	Address: Va	alue	
PLAYER	Deinten	2010060		00001010
00000 -	A Duter	3718860	: P->U	JUBBI218
0004 -	- 4 Bytes Pointon	3710004	. 2 . P	2000500
▶.0000 -	- Pointer	3918860	· P=>0	139128F0
	- 4 Butes	3918870	- 3	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
	- 4 Bytes	3918874	- 3	
	- 4 Bytes	3918878	: 16	
001C -	- 4 Bytes	391887C	: 17	
0020 -	- 4 Bytes	3918880	: 2	
⊳.0024 -	- Pointer	3918884	: P->0)3919DB0
0028 -	- 4 Bytes	3918888	: 0	
002C -	- 4 Bytes	391888C	: 4294	1967295
0030 -	- 4 Bytes	3918890	: 649	
0034 -	- 4 Bytes	3918894	: 40	
0038 -	- 4 Bytes	3918898	: 60	
003C -	- 4 Bytes	391889C	: 50	
0040 -	- 4 Bytes	39188A0	: 0	
0044 -	- 4 Bytes	39188A4	: 0	
0048 -	- 4 Bytes	39188A8	: 0	
004C 🚄	4 Bytes	39188AC	: 0	-
····0050 -	- 4 Bytes	39188B0	: 668	
0054 -	- 4 Bytes	39188B4	: 30	
	- 4 Bytes	39188B8	: 0	
	- 4 Bytes	39188BC	: U	
- 0060 -	- 4 Bytes	3918800	.: 1 	0010720
. 0064 -	- Fointer	3718864	· P->0	13718/EU 12010000
♪.0000 -	- rointer	3910000	: F->0	3310330
Active Desc	ription	Address	Туре	Value
pointe	erscan result	P->039188B0	4 Bytes	668
Auto.	Assemble scrip	t 🔪		<satipt></satipt>
Advanced Or	otions			AF.
				*

Ahora a partir de aquí es cuestión de "tocar, explorar y probar". **Mi recomendación personal es que cerca de un offset válido se encuentran el resto de offsets de la estructura.** Lo primero que ahoremos es poner un nombre a los offsets conocidos, por el momento solo **conocemos el offset +50** así que damos **doble-click encima** y le pones un nombre:

🐔 Structure Info	
Offset	
50	45
Description	
VIDA	
Туре	
4 Bytes	•
🔲 Hexadecimal 📄 Signed	
Background Color	
Structure pointed to	Offset into
Undefined	- 0
OK Cancel	

Ahora lo interesante sería poder detectar que offsets se corresponden al ataque, rango, magia, defensa, etc... estoy seguro que esos offsets estarán cerca de la vida (+50) así que miraré los offsets cercanos y los compararé con mi jugador:

Offset-description	Address: Value						
PLAYER							
⊳.0000 - Pointer	3918860 : P->00BB1218						
-0004 - 4 Bytes	3918864 : 2						
⊳0008 - Pointer	3918868 : P->0390C580						
⊳000C - Pointer	391886C : P->039128F0						
-0010 - 4 Bytes	3918870 : 3						
-0014 - 4 Bytes	3918874 : 3						
-0018 - 4 Bytes	3918878 : 16						
-001C - 4 Bytes	391887C : 17						
-0020 - 4 Bytes	3918880 : 2						
0024 - Pointer	3918884 : P->03919DB0						
-0028 - 4 Bytes	3918888 : 0						
-002C - 4 Bytes	391888C : 4294967295						
-0030 - 4 Bytes	3918890 : 649						
-0034 - 4 Bytes	3918894 : 40						
-0038 - 4 Bytes	3918898 : 60						
-003C - 4 Bytes	391889C : 50						
-0040 - 4 Bytes	39188A0 : O						
-0044 - 4 Bytes	39188A4 : O						
-0048 - 4 Bytes	39188A8 : O						
-004C - 4 Bytes	39188AC : 0						
- 0050 - VIDA -	39188B0 : 668						
.0054 - 4 Bytes	39188B4 : 30						
-0058 - 4 Bytes	39188B8 : 0						
-005C - 4 Bytes	39188BC : 0						
-0060 - 4 Bytes	39188C0 : 1						
> 0064 - Pointer	39188C4 : P->039187E0						
⊳0068 - Pointer	39188C8 : P->03918990						



Me llama la atención el offset +34... tiene un valor de 40 que coincide con los 40 puntos de daño en el juego:

⊳0024 - Pointer	3918884 : P->03	Woodland els M
-0028 - 4 Bytes	3918888 : 0	
-002C - 4 Bytes	391888C : 42949	
-0030 - 4 Bytes	3918890 : 649	🖳 🦳 🗌 Att/Dmg
-0034 - 4 Bytes	3918894 : 40	40/0
-0038 - 4 Bytes	3918898 : 60	
-003C - 4 Bytes	391889C : 50	MANJATE # 29201
-0040 - 4 Bytes	39188AO : O	
-0044 - 4 Bytes	39188A4 : O	🖉 📈 🕅 🎍 ee e
-0048 - 4 Bytes	39188A8 : O	
-004C - 4 Bytes	39188AC : 0	
-0050 - VIDA	39188B0 : 668	

Así que hacemos click derecho encima del offset +34 y seleccionamos la opción "Change Value" e introducimo un nuevo valor, por ejemplo 77



BINGO! El offset +34 es el encargado de almacenar el daño (Melee Attack) así que hacemos doble-click para editarlo y apuntamos el offset a nuestras notas:



Está clarísimo que **los offsets entre +34 y +50 serán los otros stats**, así que iremos editando cada offset para conocer su valor en el juego y lo anotaremos en nuestra disección y en nuestras notas. Así me ha quedado a mi:

Offset-description	Address: Value	
PLAYER		
⊳0000 - Pointer	3918860 : P->00BB1218	
-0004 - 4 Bytes	3918864 : 2	🔛 info.txt - NFOPad*
⊳0008 - Pointer	3918868 : P->0390C580	Archivo Editar Configuraciones Ver Avuda
⊳000C - Pointer	391886C : P->039128F0	PIDECCION DECEMP UTDA
-0010 - 4 Bytes	3918870 : 3	DIRECCION RESIAR VIDA
-0014 - 4 Bytes	3918874 : 3	
0018 - 4 Bytes	3918878 : 16	HackSlashLoot.exe+1504/4
001C - 4 Bytes	391887C : 17	DI NUED CEDUCEUDE
0020 - 4 Bytes	3918880 : 2	PLAYER SIRUCIORE
⊳0024 - Pointer	3918884 : P->03919DB0	
0028 - 4 Bytes	3918888 : 0	+50 = HP
002C - 4 Bytes	391888C : 4294967295	+40 = DEF
0030 - 4 Bytes	3918890 : 649	+34 = Melee Atk
-0034 - MELEE ATTACK	3918894 : 77	+38 = Range Atk
-0038 - RANGE ATTACK	3918898 : 60	+3C = Magic Atk
-003C - MAGIC ATTACK	391889C : 50	+44 = Melee Dmg
-0040 - DEFENSE	39188A0 : 0	+48 = Range Dmg
-0044 - MELEE DMG	39188A4 : 0	+4c = Magic Dmg
-0048 - RANGE DMG	39188A8 : 0	
-004C - MAGIC DMG	39188AC : 0	
0050 - VIDA	39188B0 : 668	
0054 – 4 Bytes	39188B4 : 30	
0058 - 4 Bytes	39188B8 : O	
005C - 4 Bytes	39188BC : 0	
-0060 - 4 Bytes	39188C0 : 1	

Bueno, ya tengo la estructura diseccionada al completo, hay otros offsets, quizás alguno esté almacenando algo interesante, pero eso ya lo veremos más adelante. Ahora **ya tenemos la estructura diseccionada con los offsets principales.** Solo nos queda poder calcular automáticamente la **base-address** de nuestra estructura para luego sumar +50 +34 +etc... e ir sacando cada una de las direcciones importantes. ¿Cómo lo hacemos? Pues **tendremos que encontrar un patrón que no sidentifique la base-address de la estructura de nuestro jugador**. Empezemos...

AOB TO MEMORY DATA

Necesitamos encontrar un patron en la memoria del programa/juego que nos identifique la base-address de forma automática. El método es muy sencillo, hay que buscar la base-address manualmente y copiar los bytes que contiene dicha base-address. Éstos pasos hay que repetirlos varias veces, dependiendo del juego y de su complejidad, necesitarás repetir ésta operación 5, 6, 7 o 10 veces. En el caso de Hack, Slash, Loot he realizado 5 búsquedas hasta encontrar un patrón válido.

Podemos empezar de varias formas, pero a mi me gusta partir siempre de la instrucción que modifica un offset de nuestra estructura. En las notas tenemos apuntado lo siguiente:

🔛 info.txt - NFOPad									
<u>A</u> rchivo <u>E</u> ditar <u>C</u> o	nfiguracion	es <u>V</u> er A <u>y</u> uda							
DIRECCION RESTA	AR VIDA		*						
HackSlashLoot.exe+150474									
PLAYER STRUCTUR	₹E								
+50 +40 +34 +38 +3c +44 +48 +4c		HP DEF Melee Atk Range Atk Magic Atk Melee Dmg Range Dmg Magic Dmg							
Lin: 1, Col: 1	C:\Users\J	ordi\Desktop\Tuto	rial∖info.txt						

Sabemos que la instrucción HackSlashLoot.exe+150474 modifica el offset +50 de nuestra estructura, así que abrimos el Memory View, hacemos click derecho y pulsamos "GoTo Address":

Goto Addre	ss		×
Fi			
HackSlashL	oot.exe+15047	4	-
	ОК	Cancel	

Estaremos delante de la instrucción famosa **SUB** que se encarga de restar la vida, así que pondremos un **breakpoint** en dicha instrucción (si pulsas **F5** se pone el **Breakpoint**) o podemos hacer click derecho y seleccionar "Toggle Break Point". Se nos quedará marcada la instrucción en **verde** indicando que hay un **breakpoint**:

		Ha	HackSlashLoot.exe+150474					
Address	Bytes	Opcode	1					
HackSlashLoot.exe+150474	29 43 50	sub	(ebx+30),eax					
HackSlashLoot.exe+150477	83 7B 50 00	cmp	dword ptr [ebx+50],00					
HackSlashLoot.exe+15047B	7F OC	<u>r</u> ig	HackSlashLoot.exe+150489					
HackSlashLoot.exe+15047D	53	push	ebx					
HackSlashLoot.exe+15047E	8B 03	mov	eax,[ebx]					
HackSlashLoot.exe+150480	FF 90 B0010000	call	dword ptr [eax+000001B0]					

Ahora volvemos al juego, **iniciamos una pelea y dejamos que el enemigo nos golpee**, verás que el juego se queda "congelado" ya que el breakpoint ha detenido la ejecución del juego así que volvemos a **CE** y nos encontraremos lo siguiente:

🗱 Memory Viewer - Currently	y debugging thread 12D4			_							
File Search View Debug Tools Kernel tools											
	HackSlashLoot.exe+150474										
Address	Bytes Opcode		Commer 🔺	EAX 00000001	CF 0						
>>HackSlashLoot.exe+150474	29 43 50 sub	(ebx+30),eax		EBX 03918860	PF 0						
HackSlashLoot.exe+150477	83 7B 50 00 cmp	dword ptr (ebx+50),00	0	ECX 00000000	AF O						
HackSlashLoot.exe+15047B	7F OC	HackSlashLoot.exe+150489		EDX 000000C	ZE 0						
HackSlashLoot.exe+15047D	53 push	ebx		ESI 03918860	SFO						
HackSlashLoot.exe+15047E	8B 03 mov	eax,[ebx]		EDI 0019D990	DEO						
HackSlashLoot.exe+150480	FF 90 B0010000 call	dword ptr [eax+000001B0]		EBP 0101FE9C	OF 0						
HackSlashLoot.exe+150486	83 C4 04 add	esp,04	4	ESP 0101FE94							
HackSlashLoot.exe+150489	A1 8C36BB00 mov	eax,[HackSlashLoot.exe+7B368C]		ETP 00550474	>						
HackSlashLoot.exe+15048E	FF 73 1C push	[ebx+1C]		Segment Registers							
HackSlashLoot.exe+150491	FF 73 18 push	[ebx+18]		Jeginent negisters	-						
HackSlashLoot.exe+150494	50 push	eax		CS 0023							
HackSlashLoot.exe+150495	88 00 mov	eax,[eax]		SS 002B							
HackSlashLoot.exe+150497	FF 50 48 call	dword ptr [eax+48]		DS 002B							
HackSlashLoot.exe+15049A	83 C4 OC add	esp,0C	12	ES 002B							
HackSlashLoot.exe+15049D	83 F8 00 cmp	eax,00	0	FS 0053							
< III			+ -	GS 002B							
	subtract										
Protect:Read/Write	Base=00601000 Size=10	00 Module=HackSlashIoot	eve .								
address 00 01 02 03	3 04 05 06 07 08 09 0A	OB OC OD OE OF 01234567	789ABCDEF	Return Address	Parameters						
00601000 00 08 00 00) 60 B2 BC 00 00 00 00	00 01 00 00 00		0055CB91	0391886						
00601010 03 00 00 00	01 00 00 00 00 00 00 00	00 00 00 00 00		00550561	0391886						
) 11 00 00 00 A0 EC BC			0055D91E	0399535						
) 45 EC BC 00 45 EC BC		• • • • • • • • • •	00539075	0019094						
00601050 06 00 00 00	0 00 00 00 00 00 00 00 00	00 00 00 00 00		00530075	0000000						
00601060 00 00 00 00) OO OO OO OO <mark>O1 OO OO</mark>	00 00 00 00 00		00000000	0000000						
00601070 00 00 00 00	0 00 00 00 00 00 00 00 00	00 00 00 00 00	• • • • • • • • • •	006004AD	0101FF4						
	J 11 UU UU UU A7 EC BC			0040151D	0000000						
) AC EC BC 00 AC EC BC			00401257	0000000						
00601080 06 00 00 00				004012C8	0000000						
006010C0 00 00 00 00	0 00 00 00 00 01 00 00	00 00 00 00 00		7630338A	7EFDE00						
006010D0 00 00 00 00	0 00 00 00 00 00 00 00	00 00 00 00 00		77009F72	7EFDE00						
006010E0 00 00 00 00) 11 00 00 00 AE EC BC	00 00 00 00 00		77009E45	HackSlas						
00601080 00 00 00 00			• • • • • • • • • •	0000000	HackSlas						
			· · · · · · · · ·	0000000	Hackalas						
00001110 00 00 00 00											

A la derecha aparece el **stack** y los **registros**. Lo que nos interesa son los registros para poder conocer la **base-address** de nuestra estructura. La instrucción es **sub [ebx+50],eax** por lo que el valor de **EBX** nos mostrará la **base-address**, en mi caso **03918860**. Pulsamos **click-derecho sobre EBX** y seleccionamos "Show in hex view":

HackSlashLoot.exe	+150474	Registers:				Flags	
Opcod	e	*	EAX	0000	0001	CF 0	
sub cmp	(ebx+31),eax dword ptr (ebx+50),00	0		EBX ECX	(201 (Show	in disassen
-ig push	HackSlashLoot.exe+150489 ebx			EDX ESI	1	Show	in hexview

Automáticamente, en la parte inferior (Hex Dump) nos mostrará la dirección de EBX (03918860), en mi caso:



Lo que tenemos que hacer ahora es **coger esos bytes y copiarlos a nuestras notas.** Yo siempre cojo **3 líneas**. Para ello con el ratón seleccionamos desde el primer byte hasta el último (se quedarán marcados en rojo) y pulsamos **Ctrl+C** para copiar, a continuación los pegamos en nuestro **fichero de texto**:

subtract																		
Protect:F address	Read 60	l∕Wr 61	rite 62	• E 63	Base 64	e=03	918 66	8000 67) Si 68	ize= 69	980 6A)00 6B	60	6D	<u>6</u> E	6F	01234567	89ABCDEF
03918860	18	12	BB	00	03	00	00	00	80	C5	90	03	FO	28	91	03		. (.
03918870	03	00	00	00	03	00	00	00	10	00	00	00	11	00	00	00		
03918880	02	00	00	00	B0	9D	91	03	00	00	00	00	\mathbf{FF}	\mathbf{FF}	\mathbf{FF}	\mathbf{FF}		
03918890	89	02	00	00	4D	00	00	00	3C	00	00	00	32	00	00	00	M	< 2
039188A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
039188B0	9C	02	00	00	1E	00	00	00	00	00	00	00	00	00	00	00		
039188C0	01	00	00	00	E0	87	91	03	90	89	91	03	20	58	91	03		. X .
039188D0	60	0D	61	00	60	0D	61	00	60	0D	61	00	10	00	00	00	`.a.`.a.	`.a
039188E0	ΔO	2C	61	00	ΑO	2C	61	00	00	00	00	00	00	00	00	00	.aa.	
039188F0	00	00	00	00	00	00	00	00	75	43	0B	BD	01	00	00	00		uC
03918900	00	00	00	00	ΑO	2C	61	00	ΑO	2C	61	00	ΑO	2C	61	00	a.	.aa.
03918910	28	00	00	00	3C	00	00	00	32	00	00	00	00	00	00	00	(2
03918920	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
03918930	00	00	00	00	10	00	00	00	50	89	91	03	ΑO	2C	61	00		P.,a.
03918940	00	00	00	00	00	00	00	00	00	00	00	00	6D	00	00	00		M
03918950	80	0C	61	00	01	00	00	00	DO	68	24	00	01	00	00	00	.a	h\$
03918960	20	00	00	00	08	00	00	00	FO	2A	91	03	AO	2C	61	00		* . ,a.
03918970	ΔO	6A	91	03	00	50	91	03	ΔO	2C	61	00	ΔO	2C	61	0.0	i P	a a

🔡 info	.txt - I	NFOR	Pad*																				x
Archiv	o <u>E</u> c	litar	<u>C</u> o	nfig	uraci	ones	<u> </u>	er /	4 <u>y</u> ud	a													
DIREC	CIO	N RE	ST	AR 1	/ID/	ł																	*
HackSlashLoot.exe+150474																							
PLAYER STRUCTURE																							
		+50		=			нр																
	-	+40		=			DEI																
	+	+34		=			Mel	Lee	At]	c													
	+	+38		=			Rar	ige	At]	c													
	+	+3c		=			Mag	gic	At]	c													
	-	+44		=			Me!	Lee	Dmg	3													
	-	+48		=			Rar	ige	Dmg	3													
	-	+4C		=			ма	JIC	Dmg	3													
18 12	BB	00	03	00	00	00	80	C5	90	03	FO	28	91	03	03	00	00	00	03	00	00	00	
10 00	00	00	11	00	00	00	02	00	00	00	В0	9D	91	03	00	00	00	00	FF	FF	FF	FF	
																							-
Lin: 17,	Col: 7	2		C:\	User	s\Jor	rdi∖D	eskto	p/T	utori	al∖in	fo.tx	:										

Esos son los bytes que se corresponden a la base-address de la estructura de nuestro personaje. Ahora hay que repetir todos éstos pasos un **mínimo de 3 veces**. Para ello pulsaremos **F9** en la ventana de **Memory View** para continuar la ejecución a partir del **Breakpoint** y luego **cerraremos por completo el juego**. Abriremos de nuevo el juego, iniciaremos una nueva partida y volveremos a seleccionar el proceso de **Hack, Slash, Loot** para trabajar con **CE**. A partir de aquí hay que repetir los pasos de éste tutorial, es decir:

- Iniciar el Memory View
- Ir a la instrucción que modifica la estructura, en nuestro caso HackSlashLoot.exe+150474
- Poner un Breakpoint
- Iniciar un combate en el juego hasta recibir daño
- Mirar los registros (EBX) y hacer "Show in hex view"
- Copiar los bytes al fichero de texto
- Cerrar el juego y volverlo a iniciar para empezar de nuevo

Estos pasos los he realizado 5-6 veces y al final he conseguido las siguientes líneas:

NFO	inf	o.b	d - 1	VFO	Pad*																				x
A	rchi	vo	Ed	litar	<u>C</u> o	nfig	uraci	ones	<u>V</u> e	er /	Ayud	la													
DI	DIRECCION RESTAR VIDA																								
	HackSlashLoot.exe+150474																								
	MUCADINGUI CACTIGUI/1																								
PI	PLAYER STRUCTURE																								
			4	+50		=			ΗР																
			+	+40		=			DEI	F															
			+	-34		=			Me1	lee	At]	k													
			+	-38		=			Rar	nge	Atl	k													
			+	+3c		=			Mag	gic	At]	c													
			+	+44		=			Me.	Lee	Dmg	3													
			1	48		_			Max	nge	Dmg	3													
				-40		-			Mag	JIC	Ding	9													
18	3 1	2	BB	00	03	00	00	00	80	C5	90	03	FO	28	91	03	03	00	00	00	03	00	00	00	
10	0 0	0	00	00	11	00	00	00	02	00	00	00	BO	9D	91	03	00	00	00	00	FF	FF	FF	FF	
18	3 1	2	BB	00	03	00	00	00	18	84	BB	00	34	84	BB	00	03	00	00	00	03	00	00	00	
21	30	0	00	00	1A	00	00	00	06	00	00	00	BC	83	BB	00	00	00	00	00	01	00	00	00	
18	3 1	2	BB	00	04	00	00	00	18	84	BB	00	34	84	BB	00	03	00	00	00	03	00	00	00	
21	3 0	0	00	00	0B	00	00	00	06	00	00	00	BC	83	BB	00	00	00	00	00	01	00	00	00	
18	3 1	2	BB	00	04	00	00	00	48	85	BB	00	34	84	BB	00	03	00	00	00	03	00	00	00	
10	C 0	0	00	00	22	00	00	00	04	00	00	00	BC	83	BB	00	00	00	00	00	FF	FF	FF	FF	
		_																							
18	31	2	BB	00	04	00	00	00	C8	84	BB	00	EO	84	BB	00	03	00	00	00	03	00	00	00	
33	5 0	0	00	00	UD	00	00	00	02	00	00	00	ыс	83	БВ	00	00	00	00	00	01	00	00	00	
18	3 1	2	BB	00	02	00	00	00	20	FD	7F	03	DO	53	80	03	03	00	00	00	03	00	00	00	
17	4 O	0	00	00	06	00	00	00	04	00	00	00	50	E5	80	03	00	00	00	00	01	00	00	00	
																									-
Lin	:1	Col	l: 1			C:\	User	s\Jor	di∖D	eskto	T/ac	utori	al\in	fo.tx	t										
	/						2221		21,12				2.1 (.11)												- 111

Ahora solo tenemos que calcular un patrón válido partiendo de éstas muestras. Se puede hacer a mano, pero para los más perezosos he programado un script en lenguaje VBS que realizará el cálculo automáticamente

El código VBS está en pastebin, solo tienes que copiarlo en un notepad y guardarlo con extensión *.vbs

```
http://pastebin.com/tQsvbSkh
Set oWSH = CreateObject("WScript.Shell")
Set oFSO = CreateObject("Scripting.FileSystemObject")
T = InputBox("Enter array of bytes n° 1:")
T = T & vbcrlf & InputBox("Enter array of bytes n° 2:")
X = 3
While MsgBox("Do you want to introduce another array of bytes?", vbYesNo, "AoB Pattern
Generator") = vbYes
       T = T & vbcrlf & InputBox("Enter array of bytes n° " & X &":")
       X = X + 1
Wend
AoB = Split(T, vbcrlf)
F = ""
W = 0
X = 0
For i = 1 To Len(AoB(0))
       For u = 1 To UBound (AoB)
               If Mid(AoB(u), i, 1) \iff Mid(AoB(0), i, 1) Then
                       F = F \& "?"
                       W = W + 1
                       X = 1
                       Exit For
               End If
       Next.
       If X \ll 1 Then F = F \& Mid(AoB(0), i, 1)
       X = 0
Next.
Set File = oFSO.CreateTextFile("aob.txt")
       File.Write "Original array of bytes:" & vbcrlf & vbcrlf
       File.Write Replace(T, vbcrlf & vbcrlf, vbcrlf) & vbcrlf & vbcrlf
       File.Write "Total array of bytes: " & UBound(AoB) + 1 & vbcrlf
       File.Write "Total wildcards used: " & W & vbcrlf & vbcrlf
       File.Write "Your AoB Pattern:" & vbcrlf & vbcrlf & F
File.Close
'MsgBox F
If MsgBox("AoB Patter Generator finished" & vbcrlf & vbcrlf & "Do you want to open
aob.txt file?", vbYesNo, "AoB Pattern Generator") = vbYes Then
       oWSH.Run "notepad.exe aob.txt"
End If
```

El script **comparará todos los bytes y buscará un patrón**. Si un byte no coincide será sustituido por el carácter **?**. Al final, mi script me dice que el Array of Bytes (**AoB**) es:

Como puedes observar, se puede calcular fácilmente a mano, solo hay que sustituir los bytes que no son iguales por un simbolo de interrogación. Ahora ya tenemos un array de bytes que identificará la base-address de la estructura. Volvemos al CE, abrimos el Memory View y nos vamos al editor de Auto-Assemble (Ctrl+A) en el menu "Tools", en el editor pegamos el siguiente código:

[ENABLE]

unregistersymbol(_player)

Solo tendrás que **sustituir el array AoB por el valor que hayas encontrado.** Una vez copiado el texto vamos al menu: "File / Asign to current cheat table" para añadir el **script** a la tabla, obtendremos lo siguiente:

Active	Description	Address	Туре	Value
	pointerscan result	P->038388D0	4 Bytes	667
	Auto Assemble script			<script></script>

Mi recomendación es que edites los nombres para no confundirte:

Active	Description	Address	Туре	Value
	pointerscan result	P->038388D0	4 Bytes	667
	Godmode			<script></script>

El **script** es muy sencillo, cuando lo activamos lanzamos un **AoB Scan** (buscar un array de bytes) y le indicamos que busque nuestro **patrón de bytes**, **CE** localizará la coincidencia y nos devolverá el resultado en la etiqueta **_player**. Eso significará que podremos utilizar la etiqueta **_player** para referirnos al **inicio de la estructura (base-address)**.

Volvemos al CE y seleccionamos "Add Address Manually"

Memo	ory View		0	Add Address Manually		
Active	Description	Address	Туре	Value		
	pointerscan result Godmode	P->038388D0	4 Bytes	667 <script></script>		

🗱 Add address	— ×
Address:	
[=???
Description	
No description	
Туре	
4 Bytes	•
Pointer	
ОК	Cancel

🐔 Add address	×
Address:	
_player	=???
Description	
BASE	
Туре	
4 Bytes	•
Pointer	
ОК	Cancel

Pulsamos **OK** y se añadirá la dirección:

Active	Description	Address	Туре	Value
	pointerscan result	P->038388D0	4 Bytes	667
	Godmode Search Player Structure BASE	(_player)	4 Bytes	<script></script>

Actualmente **CE** no reconoce la dirección **_player**, por eso no muestra nada. Ahora lo que haremos es activar el script con el **AoB** y sucederá la magia...

Active	Description	Address	Туре	Value
	pointerscan result	P->0FF3AAF0	4 Bytes	30
	Godmode Search Player Structure BASE	OFF3AAA0	4 Bytes	<script></script>

Tachan!! **CE** ha ejecutado el **AoB**, ha buscado el **patrón de bytes** y ha devuelto el resultado en la variable **_player**, ahora **CE** ya sabe el valor de dicha dirección y nos muestra **OFF3AAAO** que se corresponde al actual **base-addres** de la estructura de nuestro personaje. A partir de aquí ya tenemos el camino hecho, solo falta completarlo.

Desactivamos el script y volvemos a pulsar "Add Address Manually", añadiremos ahora la siguiente dirección:

🐔 Add address	X
Address:	
_player+50	=???
Description	
VIDA	
Туре	
4 Bytes	•
Pointer	
ОК	Cancel

La explicación es sencilla; estamos añadiendo una nueva dirección que será base-address+50 que según nuestras notas se corresponde al offset de la vida. Hacemos lo propio con los offsets encontrados:

- +50 = HP • +40 = DEF
- +34 = Melee Atk
- +38 = Range Atk
- +3c = Magic Atk
- +44 = Melee Dmg
- +48 = Range Dmg
- +4c = Magic Dmg

Obtendremos lo siguiente:

Description	Address	Туре	Value
pointerscan result	P->0FF3AAF0	4 Bytes	30
Godmode			<script></script>

Ahora que ya tenemos todos los offset introducidos solo falta activar el script AoB para recoger la estructura:

Active	Description	Address	Туре	Value
	pointerscan result	P->0FF3AAF0	4 Bytes	30
	Godmode			<script></script>

BAMP! De un solo golpe ya tenemos todos los valores. Además como estamos usando un **AoB Scan**, en la próxima ejecución del juego **CE** buscará la **base-address** y la guardará en **_player**, por lo que podremos visualizar y editar las direcciones dinámicas de la estructura de nuestro jugador.

Os recomiendo que con vuestro ratón ordenéis la tabla y dejéis todos los offsets "dentro" del script:

Active	Description	Address	Туре	Value
	pointerscan result	P->0FF3AAF0	4 Bytes	30
	Godmode Search Player Structure BASE VIDA DEFENSA MELEE ATK RANGE ATK MAGIC ATK MELEE DMG RANGE DMG MAGIC DMG	OFF3AAAO OFF3AAF0 OFF3AAE0 OFF3AAD4 OFF3AAD8 OFF3AADC OFF3AAE4 OFF3AAE8 OFF3AAEC	4 Bytes 4 Bytes 4 Bytes 4 Bytes 4 Bytes 4 Bytes 4 Bytes 4 Bytes 4 Bytes	<script></script>

Para luego hacer click derecho en el script y seleccionar:

	Active Description	Address	Туре	Value
	pointerscan resul	t P->OFF3AAF0	4 Bytes	30 <script></script>

Así los offsets permanecerán ocultos si el script está desactivado y se mostrarán cuando activemos el script.

Bueno, pues de ésta forma se consigue encontrar un **patrón de bytes** que apunte a una **estructura dinámica** para luego sacar los **offsets estáticos** de su interior. Practicad!

TELEPORT HACK

Ya que hemos llegado hasta aquí nos podemos esforzar un poco más para terminar de rizar el rizo. Os voy a explicar como utilizar el programa WinMerge para comparar 2 estructuras y buscar los offsets de forma casi automática. Primero de todo nos descargamos WinMerge de su web y lo instalamos: <u>http://winmerge.org/</u>

Nuestro objetivo es crear un **Teleport Hack**, la mayoría de juegos almacenan en la propia estructura del personaje su posición en el plano X / Y. Si se trata de un juego en **3D** tendremos un tercer eje llamado Z. En el caso de **Hack**, **Slash**, **Loot**, solo habrá **2 coordenadas** para posicionar al jugador, vámos a intentar sacarlas.

Abrimos CE, abrimos el proceso de HSL y cargamos nuestra tabla con el pointer, los scripts en Auto-Asemble y la estructura diseccionada.



Pulsamos el script "Search Player Structure", la función AoB Scan hará su trabajo y nos devolverá la estructura rellenada:

	pointerscan result	P->03799080	4 Bytes	30
	Godmode			<script></script>

Hemos activado el **script** para cargar la **base-address** de la estructura de nuestro jugador en la variable **_player**. Ahora abrimos el **Memory View** y vamos al menú "Tools / Dissect data & structures":

🐔 Structure dissect						
File	File View Structures Structure Options					
Gro	up 1					
_pla	ayer					
Offset	t-descri	ption	Address: Value			
Offset	t-descri	ption	Address: Value			
Offset	t-descri	ption	Address: Value			

En el recuadro para la **base-address** podemos poner **_player**, **CE** se encargará de interpretar la variable. Abrimos el menú "Structures" y seleccionamos la estructura que ya hicimos en el capitulo anterior:

ebp,esp esp,08	🐔 Structure dissect			
[esp],00000002	File View	Structures	Structu	
Define new st	ructure	Ctrl+N		
PLAYER			ļ	
ebp ecv [HackSlashLoot	Ottset-descri	ption		

CE nos mostrará la estructura _player con sus offsets:

🗱 Structure dissect:PLAYER						
File View Structures Structure	Options					
Group 1						
player						
Offset-description	Address: Value					
PLAYER						
⊳0000 - Pointer	3799030 : P->00BB1218					
-0004 - 4 Bytes	3799034 : 2					
⊳0008 - Pointer	3799038 : P->0378C8F0					
þ∙000C - Pointer	379903C : P->037930A0					
-0010 - 4 Bytes	3799040 : 3					
-0014 - 4 Bytes	3799044 : 3					
-0018 - 4 Bytes	3799048 : 48					
-001C - 4 Bytes	379904C : 14					
-0020 - 4 Bytes	3799050 : 2					
⊵0024 - Pointer	3799054 : P->0379A560					
0028 - 4 Bytes	3799058 : 0					
002C - 4 Bytes	379905C : 1					
0030 - 4 Bytes	3799060 : 9					
-0034 - MELEE ATTACK	3799064 : 40					
-UU38 - RANGE ATTACK	3799068 : 60					
	379906C : 50					
-0040 - DEFENSE	3799070 : 0					
-0044 - MELEE DMG	3799074 : 0					
	3799078 : 0					
	3799070 : 0					
	3799080 : 30					
	3799084 : 30					
	3/99088 : 0					
	3799080 : 0					
	3/99090 : 1					

Ahora, sin tocar nada más, abrimos el menú: "File / Save values (Ctrl+S)" y guardamos con un nombre, por ejemplo "struct_1.txt"

A continuación volvemos al juego y desplazamos nuestro personaje un par de casillas, asegurando que mueves tanto el eje X como el Y (por ejemplo, moviéndote una casilla arriba y otra a la derecha). Volvemos a la ventana de "Structure dissect" y pulsamos de nuevo menú: "File / Save values (Ctrl+S)" para guardar la estructura, pondremos otro nombre, por ejemplo "struct_2.txt"

Ahora ya tenemos 2 ficheros de texto con la misma estructura pero con el personaje en 2 posiciones diferentes. Toca hacer trabajar a WinMerge para conocer las diferencias de ambos ficheros:

差 WinMerge						
Archivo Editar Ver Herramientas Plugins Ventana Ayuda						
🗋 🕞 🕞 🗠 여 🕸 🔽 중 🖾 곳 포 Þ 🔶 🐼 🎦 🖾 🕼	Ŵ					
Seleccione archivos o carpetas						
Archivos o Carpetas a Comparar	Aceptar					
Izquierda: C: \\Tutoria \struct_1.txt Examinar	Cancelar					
Derecha: C:\\Tutoria \struct_2.txt	Avuda					
Filtro: *.* Seleccionar						
Desempaq.: Seleccionar						
Incluir subcarpetas						
Seleccione dos carpetas o archivos a comparar.						

Cargamos los 2 ficheros para comparar y pulsamos Aceptar, WinMerge nos mostrará cualquier línea que esté diferente:

0000 - Pointer 0004 - 4 Bytes 0008 - Pointer 000C - Pointer 0010 - 4 Bytes 0014 - 4 Bytes	3799030 : P->00BB1218 3799034 : 2 3799038 : P->0378C8F0 379903C : P->037930A0 3799040 : 3 3799044 : 3	0000 - Pointer 0004 - 4 Bytes 0008 - Pointer 000C - Pointer 0010 - 4 Bytes 0014 - 4 Bytes	3799030 : P->00BB1218 3799034 : 2 3799038 : P->0378C8F0 379903C : P->037930A0 3799040 : 3 3799044 : 3
0018 - 4 Bytes	3799048 : 48	0018 - 4 Bytes	3799048 : 47
001C - 4 Bytes	379904C : 14	001C - 4 Bytes	379904C : 15
0020 - 4 Bytes	3799050 : 2	0020 - 4 Bytes	3799050 : 2
0024 - Pointer	3799054 : P->0379A560	0024 - Pointer	3799054 : P->0379A560
0028 - 4 Bytes	3799058 : 0	0028 - 4 Bytes	3799058 : 0
002C - 4 Bytes	379905C : 1	002C - 4 Bytes	379905C : 1
0030 - 4 Bytes	3799060 : 9	0030 - 4 Bytes	3799060 : 9
0034 - MELEE ATTACK	3799064 : 40	0034 - MELEE ATTACK	3799064 : 40
0038 - RANGE ATTACK	3799068 : 60	0038 - RANGE ATTACK	3799068 : 60
003C - MAGIC ATTACK	379906C : 50	003C - MAGIC ATTACK	379906C : 50
0040 - DEFENSE	3799070 : 0	0040 - DEFENSE	3799070 : 0
0044 - MELEE DMG	3799074 : 0	0044 - MELEE DMG	3799074 : 0
0048 - RANGE DMG	3799078 : 0	0048 - RANGE DMG	3799078 : 0
004C - MAGIC DMG	379907C : 0	004C - MAGIC DMG	379907C : 0
0050 - VIDA	3799080 : 30	0050 - VIDA	3799080 : 30
0054 - 4 Bytes	3799084 : 30	0054 - 4 Bytes	3799084 : 30
0058 - 4 Bytes	3799088 : 0	0058 - 4 Bytes	3799088 : 0
005C - 4 Bytes	379908C : 0	005C - 4 Bytes	379908C : 0
0060 - 4 Bytes	3799090 : 1	0060 - 4 Bytes	3799090 : 1

Y aquí tenemos claramente las diferencias, offsets +18 +1C. En la primera estructura tenian el valor de 48/14 y luego han pasado a 47/15. Ahora solo queda identificar que offset es la X o la Y y crear la dirección manual como ya vimos anteriormente:

🐔 Stru	🗱 Structure dissect:PLAYER						
File	View	Structures	Structure Option	ns			
Grou	p1 -						
_play	yer						
Offset-	descrip	otion		Address: Va	lu	e	
PLAYI	ER						
⊳.00	00 -	Pointer		3799030	1	P->00BB1218	
00	04 -	4 Bytes		3799034	1	2	
⊳.00	08 -	Pointer		3799038	1	P->0378C8F0	
	UC -	Fointer		379903C	-	P->037930A0	
	10 -	4 Bytes		3799040	÷	3	
	14 -	4 Bytes		3799044	1	3	
00	10 -	-POS X (1 -POS V (1	ertical)	3799040	1	40	
	20 -	4 Bytes	creicar,	3799050	÷	2	
⊳.00	24 -	Pointer		3799054	÷	- P->0379A560	
	28 -	4 Bytes		3799058	:	0	
00	2C –	4 Bytes		379905C	÷	4294967295	
00	30 -	4 Bytes		3799060	:	9	
00	34 -	MELEE AT	TACK	3799064	1	40	
00	38 -	RANGE AT	TACK	3799068	1	60	
	3C –	MAGIC AT	TACK	379906C	1	50	
00	40 -	DEFENSE		3799070	1	0	
00	44 -	MELEE DM	IG IS	3799074	÷	0	
00	48 -	RANGE DM	IG IG	3799078	1	0	
	4C -	MAGIC DM	IG	379907C	÷	0	
	50 -	VIDA 4. Dest of -		3799080	-	30	
00	54 - E0 .	4 Bytes		3733U84 3799N00	ł	30 0	
	50 -	4 Dytes		3799080	ĵ,	0	
	60 -	4 Bytes		3799090	ĵ,	1	
1.00		. 2,000		0.0000	1	-	

Active		Description	Address	Туре	Value
		pointerscan result	P->03799080	4 Bytes	30
]	Godmode			<script></script>

Advanced Uptions

Finalmente guardamos los cambios en nuestra tabla, **activamos el script** y nos movemos por la pantalla para ver funciona. Los valores de **X/Y** se pueden editar a mano para desplazar al jugador por la pantalla. También se podría crear un **Auto-Assemble script** que almacene **la posición actual de X/Y** y luego con un **hotkey** volver a setear los offsets al valor guardado para hacer un **teleport-hack**. Pero eso ya es algo más avanzado y tengo demasiado sueño como para seguir expllicando cosas.

DESPEDIDA

Bueno, hasta aquí el tutorial, espero que os haya gustado, me ha tomado 2 días escribirlo, tomar las fotos y maquetarlo. Espero que lo disfruten y puedan probar todas y cada una de las cosas que he explicado. Nos vemos por el foro...

MadAntrax - 26/04/2014

[eof]