Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Análisis e inyección de procesos

Process analysis and injection

Joseph Gabino Rodríguez

La Laguna, 22 de mayo de 2023

Dña. **Pino Caballero Gil**, con N.I.F. 45.534.310-Z Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

D. **Javier Correa Marichal**, con N.I.F. 79.088.385-X miembro del grupo CryptULL de investigación en Criptología de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Análisis e inyección de procesos*”

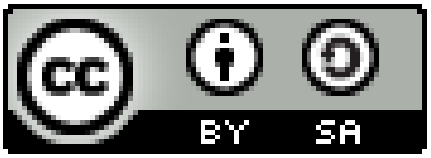
ha sido realizada bajo su dirección por D. **Joseph Gabino Rodríguez**, con N.I.F. 43.836.447-B.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 22 de mayo de 2023

Agradecimientos

En primer lugar, me gustaría expresar mi más sincero agradecimiento a mis tutores *Pino Caballero Gil* y *Javier Correa Marichal*. Su dedicación, orientación y respaldo constante fueron fundamentales para llevar a cabo este trabajo con éxito. Agradezco su paciencia y su disponibilidad, que fueron esenciales para reconducir el proyecto y superar obstáculos que iban surgiendo por el camino. También quiero agradecer a *mis compañeros de clases y profesores* por su colaboración y contribución en mi desarrollo personal y profesional. Me han brindado apoyo y consejos valioso que me permitieron enriquecer el proyecto. Por último, agradecer a mi familia y amigos por su apoyo en todo momento. Gracias a ellos, pude enfocarme en el proyecto e investigación. Quiero hacer una mención especial a mis dos mejores amigos Alejandro Lugo Fumero y Sergio Lugo Fumero, quienes me han brindado su amistad y compañía a lo largo de todo el proceso. Una vez más, quiero extender mi profundo agradecimiento a todas las personas que han contribuido de alguna manera en el desarrollo de este trabajo.

Licencia



Resumen

En la actualidad a la mayoría de usuarios de videojuegos les suele interesar hacer trampas. Esto es algo que puede verse en los diferentes mercados que están en auge, donde se paga al programador por modificar el comportamiento de los procesos. El caso más habitual es el de las plataformas como Wemod o Mrantifun.net, que no son más que entrenadores, es decir, programas que modifican el juego ofreciendo ventajas al usuario que los utilice, tanto simples procesos de almacenamiento de munición como funciones de teletransporte o vida infinita.

Por ello, en este proyecto se ha investigado cómo emular dichos comportamientos gracias a herramientas como Cheat Engine (CE). CE es un escáner de memoria en Windows que permite aplicar ingeniería inversa para modificar el comportamiento de un proceso del que no se dispone de código fuente. Permite escanear ubicaciones de memoria modificando su contenido o comportamiento mediante inyección de código, y además permite el escaneo con mapa de puntero a un nivel o multinivel. Gracias a esta herramienta se puede implementar un trainer en un juego actual, estudiando cómo maneja la memoria, qué tipos de datos utiliza para almacenar las variables y haciendo uso de las regiones cueva, que son regiones de memoria no utilizadas por el desarrollador y aprovechadas para inyectar código mediante instrucciones en ensamblador.

Palabras clave: regiones cueva, inyección, ensamblador, punteros, trucos, antitrampas, depurador, trainer

Abstract

Nowadays, most video game users are interested in cheating. This is something that can be seen in the different markets that are booming, where the programmer is paid for modifying the behavior of the processes. The most common case is that of platforms such as Wemod or Mrantifun.net, which are nothing more than trainers, i.e., programs that modify the game by offering advantages to the user who uses them, both simple storage processes of ammunition and teleportation functions or infinite life.

For this reason, this project investigates how to emulate such behaviors thanks to tools such as Cheat Engine (CE). CE is a memory scanner in Windows that allows reverse engineering to modify the behavior of a process whose source code is not available. It allows to scan memory locations modifying its content or behavior by means of code injection, and also allows the scanning with pointer map to a level or multilevel. Thanks to this tool, a trainer can be implemented in a current game, studying how it handles the memory, what types of data it uses to store the variables and making use of the cave regions, which are memory regions not used by the developer, used to inject code by means of instructions in assembler.

Keywords: cave regions, injection, assembler, pointers, cheats, anticheats, debugger, trainer

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Objetivo

El objetivo de este TFG ha sido aprender a manejar la herramienta Cheat Engine (CE) [1] para, mediante inyección, poder modificar el comportamiento de procesos. Con este fin, se propuso la división de este objetivo en 9 subobjetivos para lograr en último término implementar un *trainer* funcional en un juego actual:

1. Realizar un completo estudio del estado del arte de aplicaciones como CE o similares.
2. Analizar detalladamente la gestión de memoria en Windows.
3. Modificar variables e inyección en algunas funcionalidades del juego, tanto sobre variables locales como punteros multinivel.
4. Buscar y modificar variables no visibles en la interfaz del proceso.
5. Crear *script* en lenguaje Lua mediante CE para generar un *trainer*.
6. Estudiar el funcionamiento de los sistemas antitrampas.
7. Ver el efecto de una versión de un juego sobre un *trainer* de versiones anteriores.
8. Comparar *trainer* generado con *trainers* actuales.
9. Transformar CE para hacer un *bypass* sobre los sistemas antitrampas.

1.2. Estado del arte

Es muy frecuente que con la salida de nuevos juegos y versiones, empresas como Wemod [2] o Mrantifun [3] busquen desarrolladores que puedan cumplir con la gran demanda de desarrollo de *trainers*. A su vez, hay una creciente demanda de usuarios que quieren obtener dichos *trainers* para poder modificar los juegos. Los usuarios acuden a esas empresas para obtener dichos *trainers*. Esas aplicaciones trabajan a muy bajo nivel y además, no permiten al usuario ver su funcionamiento ya que la interfaz que proporcionan oculta su complejidad. Debido a su falta de transparencia, es muy fácil que al descargar estos *trainers* de sitios no legítimos o de forma gratuita inyecten algún tipo de *malware* que termine dañando el equipo [4]. Por este motivo, esas empresas tienen mucho éxito, ya que venden un producto de calidad que cumple tanto con las expectativas

como con los estándares de calidad esperados por los usuarios. La metodología de pago empleada en esas plataformas es mediante suscripción. En principio los usuarios tienen acceso al *trainer* pero con funciones limitadas, pudiendo solamente usarlo en ciertas versiones del juego. Si el usuario paga la cuota de suscripción, estas limitaciones desaparecen, permitiendo al usuario hacer pleno uso del *trainer*. Ese es un ejemplo de que la modificación de aplicaciones está abarcando cada vez más espacio y mercado en el sector tecnológico, llegando a más usuarios y suponiendo cada vez un problema mayor para las empresas desarrolladoras de videojuegos.

Es importante mencionar que estos *trainers* pueden tener consecuencias negativas en la experiencia de juego. A menudo, permiten a los usuarios obtener ventajas injustas en juegos multijugador, arruinando así la experiencia de los demás jugadores y creado así desequilibrios en el juego. Por otro lado, los *trainer* también pueden ser utilizados para generar modificaciones que añaden nuevas funcionalidades en el juego, lo que puede aumentar su vida útil, además de proporcionar a los usuarios una experiencia única. Sin embargo, es fundamental tener en cuenta que la modificación de juegos puede ser considerada una violación de los términos de servicio de los desarrolladores, lo que puede resultar una expulsión del juego o incluso en acciones legales.

En conclusión, la utilización de *trainers* para la modificación de juegos es una práctica cada vez más común, pero es crucial tener en cuenta los riegos asociados a la descarga ilegítima de los mismos, pudiendo acarrear graves consecuencias. Por ello es importante utilizar los *cheats* de manera responsable para evitar consecuencias negativas.

1.3. Cheat Engine

Para llevar a cabo el proyecto se ha utilizado el sistema operativo Windows y el escáner de memoria CE, desarrollado por Eric Heijnen. Esta herramienta cuenta con soporte en diferentes arquitecturas de procesadores como x86, x64, ARM y es compatible también con otros sistemas operativos como Linux y Mac OS (ver Figura 1.1).

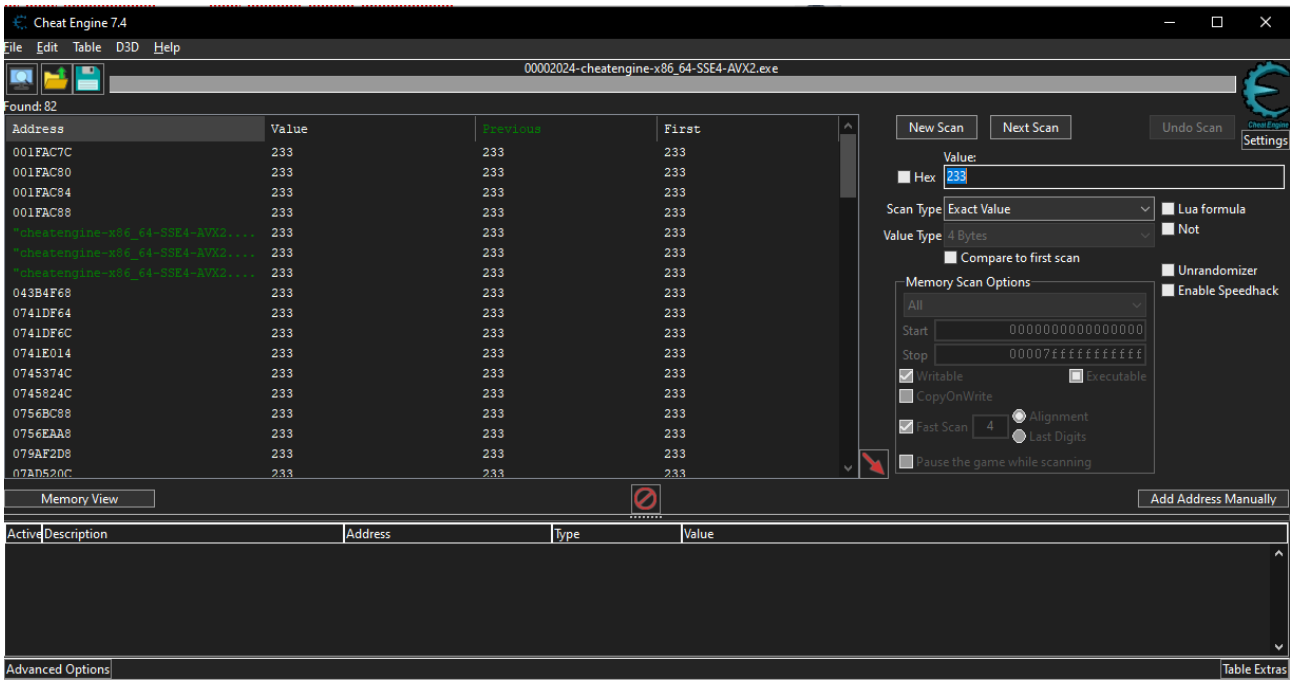


Figura 1.1: Interfaz principal de Cheat Engine

Entre algunas de las funcionalidades de CE se encuentran:

- Buscar y modificar valores en el proceso: Permite al usuario buscar y encontrar cualquier valor en el proceso anclado, como el nivel de vida o el dinero del jugador, y luego modificarlo para variar su dificultad. Dispone de más de 8 tipos de reconocimiento de datos.
- Escanear y comparar valores: Permite escanear el proceso en busca de cambios en los valores, lo que permite detectar cualquier modificación aunque se haya hecho en diferentes instantes de tiempo.
- Crear y utilizar *scripts*: Puede generar *scripts* para automatizar ciertas tareas en el juego, como aumentar el nivel de vida en cada de las acciones que modifiquen la misma. Modificando así el comportamiento del proceso.
- Herramientas de desensamblado y depuración: Además de hacer *cheats*, CE también se puede utilizar para buscar errores en el proceso, lo que permite detectar y solucionar problemas que afecten a la jugabilidad.
- Búsqueda de punteros: Tiene una herramienta para realizar un seguimiento de punteros, llegando a crear un mapa de punteros de diferentes niveles, además de almacenar registros de punteros para futuras búsquedas.
- Creación de tablas: Permite guardar las direcciones para posteriormente generar una interfaz que ejecute esas instrucciones, abstrayendo al usuario de las inyecciones necesarias para modificar el comportamiento.
- Antidepuración: Dispone de técnicas mediante el uso de puntos de ruptura para impedir que Windows lo bloquee, pudiendo llegar a trabajar con normalidad al analizar casi cualquier proceso.

La ejecución de una búsqueda sencilla en CE involucra seleccionar el proceso a analizar, buscar el valor objetivo y pulsar el botón de búsqueda. La calidad de los resultados obtenidos en la primera búsqueda puede ser significativa, alcanzando alrededor de 10 mil resultados, lo que exige utilizar patrones específicos y filtros que permitan reducir las búsquedas hasta lograr el valor objetivo. Una vez encontrado, las funcionalidades de CE pueden ser utilizadas para modificar el valor y alterar el comportamiento del proceso.

1.4. Gestión de la memoria

Para realizar inyecciones y búsquedas de valores de manera efectiva, es fundamental comprender cómo CE gestiona la memoria [5]. Por lo tanto, es esencial analizar previamente los distintos tamaños de memoria disponibles, y distinguir entre valores flotantes y enteros (ver Tabla 1.1).

Tipo valor	Byte entero	2 Byte entero	4 Byte entero	8 Byte entero	4 Byte flotante	8 Byte flotante
Nombre	Byte - 1C	Word - 1CA7	Dword	Qword	Dword	Qword
Bits	8	16	32	64	32	64
Rangos	$[0-255]$	$[0-65535]$	$[0-4294967295]$	2^{64}	$3{,}34x10^{38}$	$1{,}79x10^{308}$
Conversión	(int) - #	(int) - #	(int) - #	(int) - #	(float)	(float)
Reserva memoria	db	dw	dd	dq	dd	dq

Tabla 1.1: Tipos de datos y rango de valores en diferentes tamaños de bytes

Si se examina la Tabla 1.1, se puede observar que con 2^8 bits, se puede usar una representación sin signo de entre 0 a 255. Al considerar las dimensiones necesarias para representar un byte, se entiende que, si se quiere almacenar el valor de vida de un personaje, lo más normal sería que no requirieran más de 2 o 4 bytes, ya que los juegos normalmente manejan valores para la vida de 100, 500 o 1000. Por tanto, es esencial identificar el tipo de valor en el que se puede estar guardando, puesto que esto reducirá los tiempos de búsquedas. Además de las formas específicas para reservar memoria en la penúltima fila, también se puede ver cómo se realizan las conversiones a entero o flotante en la última fila, puesto que al realizar las inyecciones, éstas detectarán el valor por defecto en hexadecimal.

1.4.1. Registros

Después de analizar los diferentes tamaños de memoria, se procede a investigar cómo maneja CE estos tamaños [6]. Para ello se ha elaborado la Tabla 1.2.

64 bits	**32 bits**	**16 bits**	**8 bits (superior)**	**8 bits (inferior)**
rax	eax	ax	ah	al
rbx	ebx	bx	bh	bl
rcx	ecx	cx	ch	cl
rdx	edx	dx	dh	dl
rsi	esi	si	-	sil
rdi	edi	di	-	dil
r8	r8d	r8w	-	r8b
r15	r15d	r15w	-	r15b

Tabla 1.2: Registros para cada tamaño de bit

Se puede observar que hay varios registros que guardan cierta relación entre ellos. Dependiendo del prefijo que se añada a cada registro, se puede acceder a una parte de su valor. Por ejemplo, al utilizar **r[ax]**, se accede a los 64 bits del registro, mientras que al usar **e[ax]**, solo se accede a los 32 últimos bits y el resto se rellena con ceros. La mayoría de estos registros se utilizan bajo un convenio, facilitando así la implementación de los mismos cuando se trabaja con la memoria. Por ejemplo, los registros **rax** suelen utilizarse para acumular valores y los registros **rcx** como contadores. Además, se puede acceder a los últimos 16 bits que se dividen en los 8 más altos y los 8 más bajos. Cabe mencionar que no todos los registros tienen disponibles el acceso a los 8 bits superiores. En su totalidad, estos registros suelen emplearse para guardar o transferir valores y hacer operaciones con enteros. Para ilustrar el uso de los registros de diferentes tamaños, se aplican a la dirección **0x1234567890ABCDEF** (ver Tabla 1.3).

Registro	Dirección	Bits
rax	0x1234567890ABCDEF	64
eax	0x90ABCDEF	32
ax	0xCDEF	16
ah	0xCD	8
al	0xEF	8

Tabla 1.3: Accesos a una dirección mediante los diferentes registros

Además de los registros enteros, también están los flotantes, que suelen tener la forma **xmmX**, donde X es un número que identifica un registro específico. En total, hay 16 registros disponibles de este tipo. Normalmente, estos registros se suelen utilizar para manejar operaciones de punto flotante, como cálculos y manipulación de valores con decimales. Estos registros son parte de la arquitectura de la CPU y están diseñados para proporcionar un rendimiento óptimo en estas operaciones. Al aprovechar estos registros, los programas pueden realizar cálculos más eficientes y precisos [7].

1.4.2. Flujo entre CPU y memoria RAM

Las instrucciones que maneja CE son las básicas de cualquier ensamblador [8], incluyendo operaciones de suma, desplazamiento, saltos condicionales y comparaciones. A medida que se vayan realizando los *scripts* se va profundizando en las características específicas de cada uno. Sin embargo, es fundamental comprender cómo se manejan los registros y operaciones en memoria, divididas en dos partes:

- La CPU, componente principal de un ordenador, es responsable de efectuar todas las operaciones y cálculos para que el sistema funcione correctamente.
- La Memoria RAM, tipo de memoria de acceso aleatorio, se emplea para almacenar los datos y programas que están siendo usados actualmente por la CPU.

En el contexto de las inyecciones de código, es fundamental comprender cómo se realizan los accesos a memoria dentro de la ejecución de un proceso. Existen dos tipos de operaciones en memoria: la lectura y la escritura. En el caso de la escritura, se modifica el valor almacenado en la memoria RAM, es decir, aquellos valores que se cambian para modificar el proceso. Para entender mejor el proceso de escritura en memoria, se puede observar el siguiente ejemplo: **mov [rdx+A8],rdx**. En esta operación se está accediendo a un valor de la memoria mediante el operador corchete y luego se está asignando el valor de un registro **rdx**, que estaba en la CPU. Por otro lado, en la operación **mov rdx,[rdx+A8]**, se está accediendo únicamente para almacenar un valor en un registro de la CPU (en este caso, **rdx**).

Es importante distinguir entre estos dos tipos de accesos a memoria, ya que permite optimizar los tiempos de búsqueda en CE. Se puede seleccionar qué direcciones de memoria se analizan dependiendo del acceso necesario que se esté buscando. De esta manera se pueden manipular los valores almacenados en la memoria RAM y lograr las metas propuestas dentro del proceso [9].

1.4.3. Acciones con punteros

Se ha indicado que los registros almacenan valores, pero las direcciones almacenan la información de la instrucción. Por ello, cuando se están realizando las búsquedas es imprescindible distinguir direcciones estáticas y dinámicas:

- Estática: Aunque se cierre el proceso y se vuelva a iniciar, el valor donde se almacena la variable no cambia.
- Dinámica: Si se cierra el proceso, el valor se almacena en otras direcciones para futuras iteraciones.

Eso es debido a que no siempre se tienen reservadas las mismas direcciones de memoria, ya que la memoria RAM puede estar más o menos ocupada dependiendo de la cantidad de procesos que se estén ejecutando en ese instante de tiempo. Como el objetivo es generar un *trainer* funcional independiente del espacio de memoria, es necesario centrarse principalmente en las direcciones estáticas, puesto que estas no varían de una ejecución a otra. Para conseguir una dirección estática, a menudo se debe partir de una dirección dinámica.

Los punteros son herramientas útiles en este proceso, y CE proporciona herramientas para crear mapas de punteros. A través de varias iteraciones, se puede llegar a la dirección de memoria correcta para cualquier ejecución del proceso. Estas direcciones estáticas son independientes del equipo en el que se esté ejecutando, pero no siempre es posible llegar a ellas.

1.4.4. Kernel

Es importante comprender el funcionamiento del depurador dentro de CE y cómo consigue acceder a la memoria de cualquier proceso, llegando a no ser detectado por la mayoría de antivirus o propios procesos anclados por este. Cuenta con 3 tipos de depurador muy bien diferenciados (ver Figura 1.2).

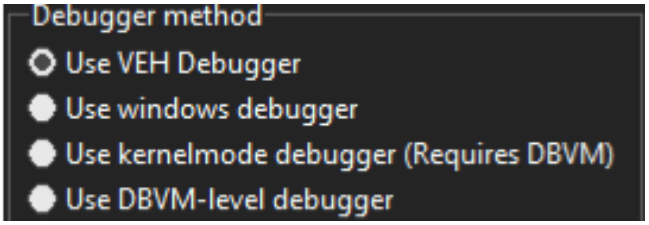


Figura 1.2: Opciones de depuración

- Vectored Exception Handling Debugger: Es una técnica que utiliza excepciones de Windows para interceptar la ejecución de programas.
- Window Debugger: Se trata del depurador interno de CE, que permite monitorear la ventana de un proceso.
- Kernelmode Debugger: Esta técnica permite depurar procesos de sistemas en anillo 0. Los usuarios pueden establecer puntos de interrupción en el código *kernel* y examinar el estado de la memoria, además de los registros del procesador. Para

hacer uso de esta técnica se debe tener activada la manipulación directa de objetos del núcleo, para poder acceder de manera más segura y controlada a estas capas del sistema.

Entre los posibles modos en los que puede funcionar CE cabe destacar el modo anillo 0, es decir, el modo más privilegiado en el modelo de núcleo de protección de memoria de un sistema operativo (ver Figura 1.3).

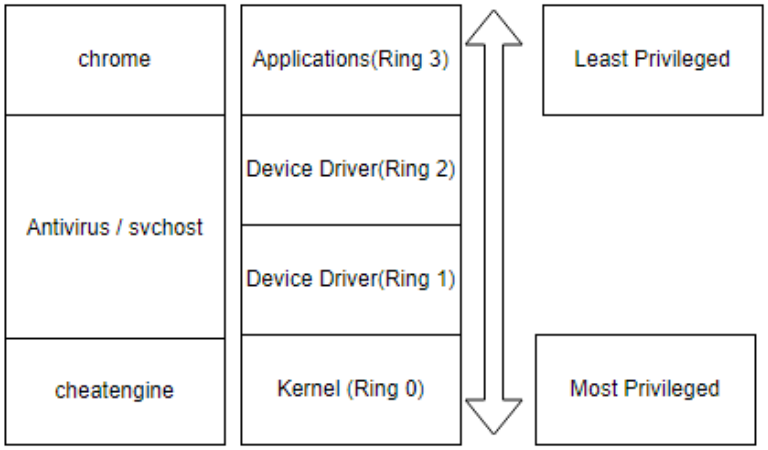


Figura 1.3: Niveles de acceso

En este modo se puede acceder por completo a todas las funciones y recursos del sistema, incluida la memoria. Para ello hace uso de controladores del sistema accediendo de esta manera a las instrucciones en ensamblador de los procesos que se están anclando. Sin embargo, esto también significa que CE tiene mayor potencial para causar daños al sistema, ya que aunque el núcleo se encargue de controlar los accesos de los procesos, garantizando que accedan de manera segura y controlada, CE puede vulnerarlos.

Es importante destacar que los antivirus utilizan técnicas similares a las de CE para detectar y eliminar software malicioso. Sin embargo, debido a que los antivirus operan en modo anillo 3, su nivel de acceso es mucho más limitado, y por lo tanto, aunque estén diseñados para detectar actividad maliciosa, no son capaces de detectar todas las formas de *cheats* que se pueden realizar con CE. Esto convierte a CE en una herramienta muy poderosa y, al mismo tiempo, muy peligrosa [10].

Capítulo 2

Inyecciones iniciales

2.1. Puesta en marcha

Una vez se han comprendido los conceptos básicos de instrucciones en ensamblador, registros, manejo de memoria, se puede proceder a realizar la inyección en el juego Nioh [11]. Este juego ha sido seleccionado debido a su clasificación de principiante en la inyección de código para manipular su comportamiento. Nioh es un juego de aventuras épicas que combina combate desafiante con elementos de *Role-Playing Game* (RPG) y una ambientación basada en la cultura samurái y la mitología japonesa. Entre algunas posibles características para inyección se incluye la vida, que representa la cantidad de daño que el personaje puede recibir antes de morir; la estamina, que controla la capacidad del personaje para realizar acciones físicas; las habilidades, que describen las capacidades especiales del personaje; el nivel, que representa el progreso a lo largo del juego; y los multiplicadores de defensa y daño, que aumentan o disminuyen la cantidad de daño que el personaje inflige o recibe tanto hacia el mismo como hacia los demás personajes.

Para comenzar el proceso de inyección, se debe generar una tabla con diferentes *scripts*, conocida como Cheat Table. En esta tabla se incluyen los códigos para modificar el comportamiento del juego, incluyendo direcciones de memoria, valores a modificar, y otras instrucciones en ensamblador que permiten la manipulación de las características.

2.2. Primera inyección

Al iniciar cualquier juego, es común que el usuario se familiarice con el ecosistema y aprenda a manipular el personaje. En este caso particular, se comenzó la búsqueda de un valor inicial que se puede encontrar en la interfaz correspondiente al dinero del personaje, que en este caso era de 1.001.750. El enfoque más lógico es realizar un escaneo de 4 bytes, considerando los bytes inferiores. Para ello, mediante una única búsqueda se obtiene el valor de una dirección estática remarcada de color verde en la interfaz de CE. Estas direcciones son las más interesantes, puesto que si se cierra el proceso y se vuelve a abrir, al realizar nuevamente la búsqueda, volverá a dar la misma dirección **nioh.exe+2176568**, por tanto, se ha acertado con el tamaño del tipo de dato (ver Tabla 2.1). Si se modifica el valor almacenado en esa dirección, se reinicia la partida, pero no el proceso, y como resultado se puede observar que el cambio se ha efectuado con éxito. Una vez se tiene la dirección de memoria, se puede ejecutar una depuración para comprobar qué instrucción accede a esa dirección. Para ello se necesita alterar el valor del dinero en el juego, ya que se necesita la ejecución de esa instrucción.

```
nioh.exe+4B07D8:
7FF7B50507CF - 48 0F4F D8  - cmovg rbx,rax
7FF7B50507D3 - E8 8061B8FF - call nioh.exe+36958
7FF7B50507D8 - 48 89 5E 08  - mov [rsi+08],rbx <<
7FF7B50507DC - 66 83 7F 0C 00 - cmp word ptr [rdi+0C],00
7FF7B50507E1 - 74 26 - je nioh.exe+4B0809
```

Registro	Hexadecimal	Decimal
rbx	00000000000F481C	1.001.500
rsi	00007FF7B6D16560	140.701.900.825.952

Tabla 2.1: Valores de los registros rbx y rsi

Se puede asumir que el valor almacenado en el registro **rbx** es el dinero que está en la CPU y dicho valor se guarda en una dirección de memoria RAM. Resulta que esa dirección([7FF7B6D16560+08]) es la dirección inicial que se encontraba en el buscador. Por tanto, es una instrucción de escritura en la memoria que actualiza un valor y por ello es un buen lugar para realizar la primera inyección. En este caso se puede simplemente inutilizar la instrucción para verificar que efectivamente la conclusión es correcta.

Tras repetir la operación con el jugador para reducir el dinero, se puede visualizar que no hay ningún tipo de reducción. Por ello se habilitan dos secciones en la inyección para la ejecución, tanto cuando el *script* está activado como cuando no lo está. Además, se añade la respectiva plantilla [12] que permite el flujo normal de ejecución (ver Figura 2.1).

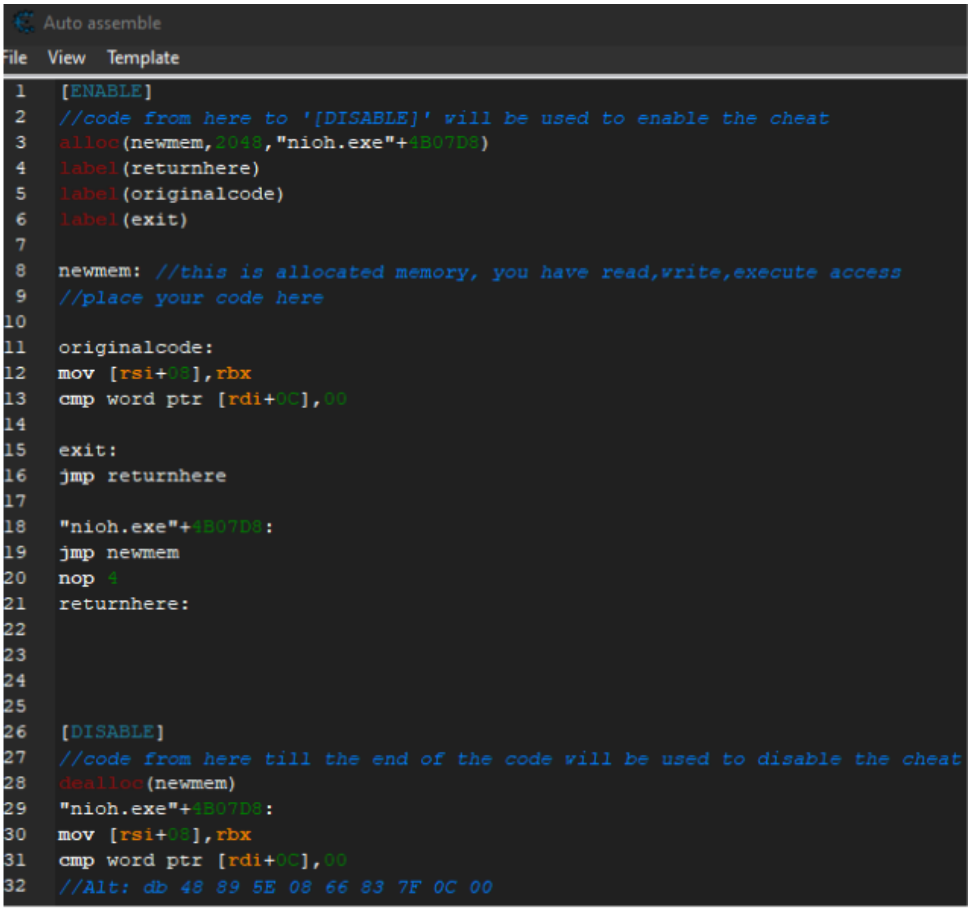


Figura 2.1: Estructura básica de inyección basada en direcciones

En la sección **enable**, se puede observar una instrucción que guarda un bloque de 2048 bits lo más cercano a la instrucción original. Esto es posible, ya que existen las denominadas regiones cuevas, regiones de memoria que el proceso no utiliza. Además, de reservar el espacio de memoria, es necesario declarar varias etiquetas usando la instrucción **label**, puesto que se necesita declarar saltos hacia direcciones externas que permitan continuar el flujo normal de ejecución para el proceso. La zona **newmen** permite escribir el código necesario, como por ejemplo una instrucción que almacene en **[rsi+08]** un entero de mil y luego hacer un salto que continúe la ejecución. Es importante tener una sección **disable** que libere región de memoria que se ha guardado, ya que de lo contrario se podría corromper la memoria y afectar a futuras inyecciones. De esta manera, cuando el *script* está activado y se realice cualquier acción que acceda a esa dirección, la cantidad de dinero aumenta automáticamente.

2.3. Modificación de estructuras

Como ya se tiene un control del dinero de los personajes, es posible proceder a modificar el nivel de cada habilidad en el juego. Para lograrlo, se hace una búsqueda del nivel actual en alguna de las habilidades, y en este caso se visualiza el valor 6. Sin embargo, se encuentra un problema, ya que hay unos 650.000 resultados donde la memoria tiene valor 6 para ese instante de tiempo. Para depurarlo y reducir las búsquedas, se va aumentando el valor dentro del juego y por cada cambio se actualiza la búsqueda. Finalmente, se encuentran 12 posibles direcciones, y entre ellas, una dirección estática **nioh.exe+2224B2C** (ver Tabla 2.2).

```
nioh.exe+4B6831:
7FF7B5056829 - 48 8B CF  - mov rcx,rdi
7FF7B505682C - E8 2701B8FF - call nioh.exe+36958
7FF7B5056831 - 89 B3 CCE50A00  - mov [rbx+000AE5CC],esi <<
7FF7B5056837 - 66 83 7F 0C 00 - cmp word ptr [rdi+0C],00
7FF7B505683C - 74 26 - je nioh.exe+4B6864
```

Registro	Hexadecimal	Decimal
rbx	00007FF7B6D16560	140.701.900.825.952
rsi	0000000000000009	9

Tabla 2.2: Valores de los registros rbx y rsi

Se observa que por cada acción que se realiza en el menú de habilidad se ejecuta 4 veces la instrucción. Por tanto, es una instrucción que se ejecuta constantemente y accede únicamente a una dirección. Además, en ese instante el registro **[e]si** (4 bytes) tiene el valor 9, que se corresponde al nivel actual de habilidad del personaje. Esta instrucción es de escritura y la dirección a la que accede es **[x7FF7B6D16560 + xAE5CC] = x7FF7B6DC4B2]**, que es la dirección estática que se estaba analizando. Para modificar la dirección de memoria, se ejecuta una inyección que cambia el valor de **esi** antes de la escritura. Para establecer el valor máximo de habilidad, se utiliza la instrucción **mov esi,(int)200**, lo que permite aumentar automáticamente la habilidad máxima del personaje al aumentar la habilidad en 1.

Se puede observar que para las 8 habilidades, comparten la misma dirección base y lo único que cambia es el desplazamiento. Esto se puede apreciar en la Tabla 2.3, donde se muestran los valores correspondientes.

Habilidad	Cuerpo	Corazón	Resistencia	Fuerza	Habilidad	Destreza	Magia	Espíritu
Desplazamiento	AE5C8	AE5CC	AE5D0	AE5D4	AE5D8	AE5DC	AE5E0	AE5E4

Tabla 2.3: Estructuras de habilidades del personaje

En la Figura 2.2 puede verse la estructura desglosada para esas habilidades. Además de observar los diferentes niveles de las habilidades, también se puede detectar el nivel total del personaje. Por tanto, existe la posibilidad de crear un *script* único que afecte a todas las direcciones, pero como se tienen directamente las direccione base, no haría falta implementarlo.

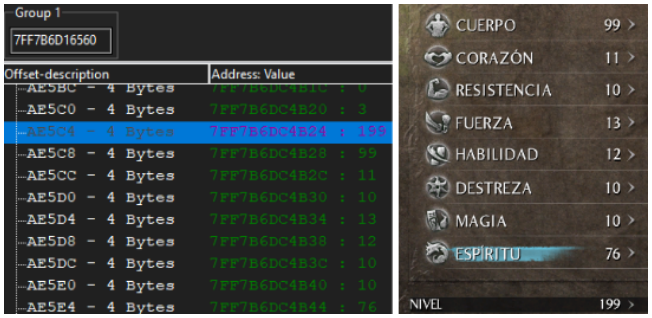


Figura 2.2: Habilidades del personaje

2.4. Código compartido

Se procede a realizar inyecciones que afecten al inventario del personaje, centrándose en la manipulación de objetos, específicamente flechas. Esto es debido al que el resto de objetos son únicos. Como punto de partida, se requiere una cantidad mínima de flechas para realizar pruebas de manera cómoda, por lo que se consigue un total de 12 unidades. A continuación, se realiza una búsqueda para valores de 4 bytes y luego de 3 iteraciones se consigue una dirección estática, tal y como se hizo en secciones anteriores. Al modificar el valor desde el propio CE, se observa que no se producen reinicios en el proceso. Posteriormente, se busca que instrucción acceden a esa dirección mediante el anclaje al proceso (ver Tabla 2.4).

```
nioh.exe+4A9338:
7FF60E5A9330 - 49 8B C9  - mov rcx,r9
7FF60E5A9333 - E8 082DE800 - call nioh.exe+132C040
7FF60E5A9338 - 66 44 29 BD 18020000  - sub [rbp+00000218],r15w <<
7FF60E5A9340 - 66 44 89 7C 24 28  - mov [rsp+28],r15w
7FF60E5A9346 - 75 1D - jne nioh.exe+4A9365
```

Registro	**Hexadecimal**	**Decimal**
r15	0000000000000001	1
rsbp	00007FF610276D80	140.694.809.701.760

Tabla 2.4: Valores de los registros r15 y rsbp

Se observa que se ejecuta una resta del valor de registro **r15[w]**, accediendo a los 8 bits más bajos. La instrucción es lógica, ya que cada vez que se lanza una flecha, la

cantidad se reduce en 1. Se procedió a realizar una inyección para inutilizar la resta, sumando un valor mayor al que se substrae, eliminando la instrucción o modificando el valor almacenado en el registro.

Una vez neutralizada la resta, se comprobó que el comportamiento del juego seguía siendo el mismo y luego de ir a la tienda a vender una cantidad de flechas se notó que la cantidad aumentaba en vez de disminuir. Por ello, como solo se ha modificado esa instrucción, se evidencia que al vender un objeto en la tienda, también se realizan llamadas a la misma; por tanto, el programador que hizo la lógica seguramente esté reutilizando la función que maneja la cantidad. Se procedió a analizar las direcciones que ejecutan esa instrucción y resulta ser que cada objeto de la tienda utiliza esa instrucción. Por tanto, se procedió a realizar una comparación entre las estructuras de las flechas originales y el resto de objetos en el inventario, buscando algún tipo de valor que los diferenciase a ambos antes de que se ejecute la lógica para el manejo de cantidades.

En CE es posible marcar instrucciones para buscar diferencias. Esta función permite identificar qué cambios se producen en memoria cuando se ejecutan ciertas acciones. En la Figura 2.3 se muestra cómo se utiliza la función mediante la interfaz de CE.

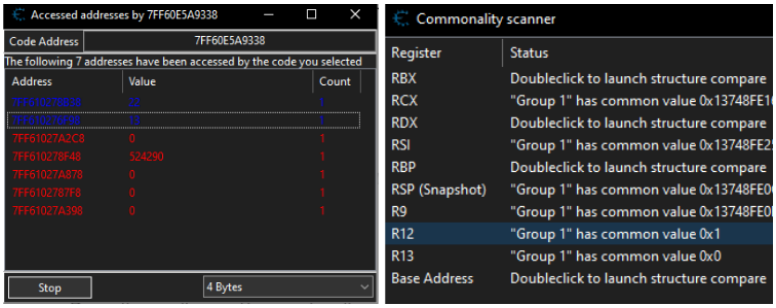


Figura 2.3: Comparación de registros

Se puede observar que cuando se ejecuta la instrucción, el registro **r12** contiene el valor 1 para el grupo de flechas en el inventario, mientras que para el resto de entidades tiene un valor distinto. Esta comparación también podría realizarse manualmente verificando el estado de los registros en el momento de la ejecución de la instrucción. Una vez se han identificado los campos que permiten diferenciar ambas estructuras, se debe añadir una instrucción de comparación en el *script*. Esta instrucción debe ser capaz de identificar cuál es el tipo de objeto que se está manipulando en cada momento.

```
cmp r12,0
je originalcode
mov r15w,#-2
originalcode:
sub [rbp+00000218],r15w
```

En este caso se ha decidido únicamente aumentar las flechas, pero sabiendo que las instrucciones son compartidas se podría emplear para tener cualquier entidad dentro del juego de manera ilimitada.

2.5. Valores flotantes

A continuación, se procedió a generar un *script* para modificar la estamina del personaje. Es un valor que se puede observar en la interfaz, por tanto, se busca el valor inicial de 4 bytes dando como salida varios resultados. Tras varias iteraciones se encontró con que

hay 0 coincidencias. Seguramente el valor donde esté almacenando la estamina no sea entero, y en consecuencia no se puede tener en cuenta lo que se visualiza en la interfaz. Por ello se procedió a repetir el proceso, pero buscando valores en flotante.

La característica especial de este valor es que se restaura con el tiempo, por tanto, se necesita detener el proceso y realizar una búsqueda en memoria. Esta, no se realiza por valores idénticos al valor original. Será necesario entonces sincronizar los movimientos del personaje con las búsquedas realizadas, con el fin de que el cambio de valor de la estamina sea predecible en cada momento. Tras más iteraciones que las de los anteriores casos, quedan 3 posibles direcciones, todas ellas dinámicas. Por tanto, se escogió la primera y se activó el depurador para analizar qué instrucciones acceden a esa dirección (ver Tabla 2.5).

```
nioh.exe+2A8A3E:
7FF60E3A8A36 - 0F28 D4  - movaps xmm2,xmm4
7FF60E3A8A39 - F3 0F10 59 38  - movss xmm3,[rcx+38]
7FF60E3A8A3E - F3 0F11 11  - movss [rcx],xmm2 <<
7FF60E3A8A42 - 0F2F D9  - comiss xmm3,xmm1
7FF60E3A8A45 - 76 5A - jna nioh.exe+2A8AA1
```

Registro	Hexadecimal	Decimal/Flotante
rxc	0000015557C00770	1.466.056.050.544
xmm2	0000000042F58000	116,0

Tabla 2.5: Valores de los registros rcx y xmm2

Destaca que se están usando instrucciones en flotante, **movss** y registros en flotante, una característica de estos registros es que solo permiten almacenar valores de otros registro o de direcciones de memoria. Por tanto, para actualizar el valor de la estamina se procede a utilizar la función **label** para declarar un valor de 4 bytes de tipo flotante.

```
alloc(auxStamina,4,"nioh.exe"+2A8A3E)
auxStamina:
  dd (float)9999.99
```

Para referirse al valor en la inyección del código, se puede utilizar el operador corchete, al igual que en el caso anterior. Sin embargo, también se encontró con el código compartido, por ello se volverá a diseccionar la estructura del jugador y la de las demás entidades (ver Figura 2.4). En vez de emplear los registros se usan los desplazamientos, ya que los registros mantienen los mismos valores.

Figura 2.4: Estado de los desplazamientos

En la dirección **rcx+78** se encuentra almacenado el valor para los jugadores representado en azul como 1, mientras que para las otras entidades el valor es 0. Con esta información, se pueden diferenciar ambas estructuras utilizando una instrucción de comparación. Cabe destacar que al poder distinguir entre ambos se puede establecer el valor

de estamina para las otras entidades a 0 haciendo que no puedan atacar al jugador, esta solución representa una ventaja adicional, ya que con una única inyección se han obtenido dos modificaciones sobre el comportamiento original del juego.

Es importante tener en cuenta que, una vez que se ha reservado espacio para almacenar el valor de estamina, es fundamental liberar ese espacio cuando ya no se necesite. Si no se libera, ese espacio quedará ocupado en la memoria y no se podrá utilizar para otras variables o datos, lo que puede llevar a problemas de memoria insuficiente y errores en la ejecución del programa. Por lo tanto, siempre es recomendable liberar la memoria reservada una vez que ya no se necesita para evitar problemas y garantizar el correcto funcionamiento del proceso.

2.6. Multiplicadores de valores

Para realizar la inyección de la vida se realizó una búsqueda basándose en la interfaz del proceso, filtrando por valores de 4 bytes. Tras varias iteraciones, se obtuvo la dirección dinámica **0x22F3CE82750**, la cual se modificó para alterar la cantidad de vida del personaje. De esta forma, se logró obtener el valor correcto, ya que la cantidad se modificaba de manera sincronizada con la interfaz (ver Tabla 2.6).

```
nioh.exe+2A050A:
7FF6E6940500 - 48 8B 43 08  - mov rax,[rbx+08]
7FF6E6940504 - C7 03 00000000 - mov [rbx],00000000
7FF6E694050A - 48 89 4B 10  - mov [rbx+10],rcx <<
7FF6E694050E - 48 3B C8  - cmp rcx,rax
7FF6E6940511 - 76 04 - jna nioh.exe+2A0517
```

Registro	Hexadecimal	Decimal
rbx	0000022F3CE82740	2.401.908.565.824
rcx	0000000000000BF0	3.056

Tabla 2.6: Valores de los registros rbx y rcx

Se puede inferir que en la dirección **0x22F3CE82740+10** se almacena el valor de la vida, pero como en ocasiones anteriores se vuelve a tener el problema del código compartido. Tras buscar entre los diferentes desplazamientos se encontró que **[rbx+A8]** en el caso de los jugadores tiene valor de 1 y 0 en otros casos. Además, el desplazamiento 10 contiene la vida máxima del personaje con el nivel actual.

Finalmente, se procedió a crear un *script* que permita que la vida de las otras entidades se reduzca a 0 después de ser golpeada. Sin embargo, se observó que, en el caso del enemigo, se requería más de un golpe para eliminarlo por completo (ver Figura 2.5).

Tras analizar las instrucciones cercanas a la inyección de vida y realizar varios intentos, se decidió utilizar el depurador para obtener más información. Se descubrió que, tras ejecutar la inyección, al volver al flujo de ejecución esperado se ejecutaba una comparación para valores menores o iguales que 0 (ver Figura 2.6). Si se cumplía la condición, se saltaba a una instrucción que neutralizaba a la entidad. Esto explica por qué en el caso del enemigo se necesitaba otro golpe para neutralizarlo. Para solucionar el problema, se necesita trasladar la inyección para que la comprobación de salto se ejecute después de la nueva inyección, de esta forma se asegura el cambio antes de la comprobación de salto condicional.

Figura 2.5: Vida actual del enemigo

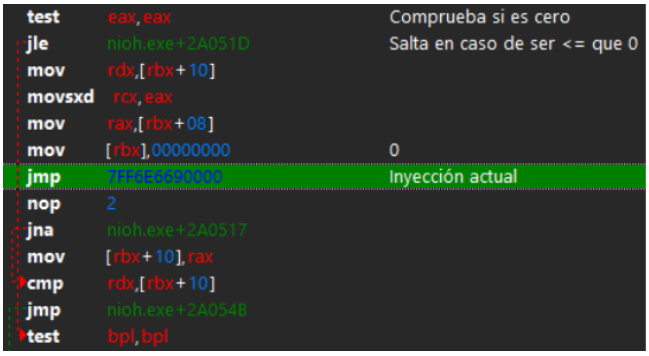


Figura 2.6: Anclaje en la inyección de la vida

Tras analizar las instrucciones superiores al salto se puede observar una operación **sub** y al comprobar con la pila de llamadas los valores, se observa que se relaciona el daño que hace el enemigo con la vida actual del personaje. En consecuencia se traslada el código a dicha instrucción. Con esto ya se tendría el flujo esperado para la inyección. Además, como se sabe que en esa dirección está el daño que recibe el jugador, se podrá proceder a realizar un multiplicador de daño/defensa para que el usuario a través de una entrada puede especificar la cantidad de daño que quiere realizar o reducir.

Se registrará una variable en la Cheat Table gracias a la función **register symbol** además de reservar el espacio de 4 bytes iniciándolo por defecto a 1. Importante cuando se desactive el *script* liberar esas secciones de memoria. Se usará la instrucción de multiplicación para enteros **imul edi,[multiplicar]**, únicamente se ejecutará este código luego de haber depurado cuando se tratan direcciones del jugador o del enemigo.

```
 ; Acceso a lectura y escritura de memoria.
newmem:
   ; Se comprueba si es jugador (Jugador[rbx+A8]=1 Enemgio[rbx+A8]=0).
  cmp [rbx+A8],1
  ; Si no es jugador salta a la instrucción original.
  jne originalcode
  ; JUGADOR
  sub eax,edi                          ; Resta a la vida el daño de ataque.
  mov eax,[rbx+8]                      ; Actualiza la vida a vida máxima.
  test eax,eax                         ; Comprueba si es menor o igual a 0.
  jle nioh.exe+2A051D                  ; Si lo es salta a instrucción de muerte.
  jmp exit
```

```
; ENEMIGOS
originalcode:
  ; Multiplica el daño de ataque por el valor de multiplicar.
  imul edi,[multiplicar]
  sub eax,edi
  test eax,eax
  jle nioh.exe+2A051D
```

A continuación se realizaron los mismos pasos para la instrucción de división en enteros **idiv**. En este caso la instrucción tiene mayor complejidad de la esperada, ya que viendo la documentación [8] se puede observar que solo recibe un operando, no como en los anteriores caso que eran 2. Se caracteriza por tener que preparar los registros de la operación antes de llamar a la misma y utiliza los registros propios del juego, por tanto, se va a necesitar declarar varias variables que almacenen temporalmente los valores de esos registros, ya que si no se perderán y posiblemente reinicien el proceso. Dependiendo del tamaño de los operandos, los valores se almacenan en unos registros específicos, a la vez que el propio cociente y resto. En este caso se quiere dividir el daño del enemigo que se está almacenando en un registro llamado **[e]di**. Por tanto, debido a su capacidad de 32 bits, la división para estos tamaños utiliza los registros **edx** y **eax** almacenando el cociente y el resto en estos.

```
    DañoAlJugador:
      ; JUGADOR
      ; Guardar el valor actual de los registros en variables temporales.
      mov [sRDX],rdx
      mov [sRAX],rax
      mov [sRCX],rcx
     ; La variable de 4byte([e]di 4byte) por tanto accede en 32bits(mode hardcode)
     ; Utilizar(edx:eax/ecx) (eax:cociente,edx:resto).
      mov edx,#0           ; 0:eax/ecx
      mov eax,edi          ; 0:DAMAGE/ecx
      mov ecx,[dividir]    ; 0:DAMAGE/5
      ; Ejecuta la división de eax por ecx, el resultado queda en eax(cociente) y edx(
    resto).
      idiv ecx
      ; Actualiza el daño de ataque con el cociente de la división.
      mov edi,eax
      ; Restaura los valores de los registros.
      mov rdx,[sRDX]
      mov rax,[sRAX]
      mov rcx,[sRCX]
```

Merece mencionar que la ejecución tiene que realizarse en la sección del jugador, ya que se quiere aumentar el daño que hace al resto de las entidades. Además, se ha empleado la función **alloc** para reservar los espacios temporales de los registros.

Finalmente, se generó un *script* que se podrá encontrar en los anexos y recoge la mayoría de los *cheats* explicados en anteriores secciones. Además, todos los scripts desarrollados podrán encontrarse en un repositorio de GitHub [13].

2.7. Interfaz del trainer

Una vez desarrollados los *cheats* para este primer juego, se ha podido observar que se le considera principiante, no porque sea fácil ejecutar las inyecciones, sino porque

se componen de varios casos particulares. Para cada una de las inyecciones se han detectado direcciones estáticas, dinámicas, código compartido, uso del depurador por comportamientos inesperados, incompatibilidades entre valores en la interfaz y valores reales.

La idea general sería asociar las inyecciones a teclas y tener esto en una interfaz para la cual el usuario no necesite ningún tipo de programa externo. Para ello se asociaron los *scripts* y mediante la interfaz de CE para manipular los formularios se añadieron apartados de descripción básica, además de un sonido cuando se utilizan. Se crearon 2 eventos para las entradas que se encargan de buscar el símbolo en memoria tanto del multiplicador como del divisor de daño y defensa, para ello primero se necesita conseguir las direcciones del proceso haciendo uso de la función **getAutoAttachList().add("nioh.exe")** .Una vez se tienen, se deben encontrar los símbolos que se registran en los propios *scripts* haciendo uso de **getAddress("multiplicar")**, y tras tener la dirección estática del símbolo se procede a obtener la propiedad **Text** asociada a la entrada del *cheat*. Solo quedaría actualizar el valor de esa dirección en memoria.

```
function CETrainer_CEEdit2Exit(sender)
 local symbol = getAddress("multiplicar")
 local value = readInteger(symbol)
 local valueInput = getProperty(CETrainer_CEEdit2,"Text")
 writeInteger(symbol, valueInput)
end
```

Es importante mencionar que al inicio del proceso de exportación se permite elegir entre dos opciones: Gigantic y Tiny. La primera opción permite descargar el ejecutable sin necesidad de disponer de CE, mientras que en la segunda si requiere que esté instalado en el sistema. Durante el proceso de creación del *trainer*, se presentaron algunos problemas, ya que el antivirus detectaba el ejecutable como un **riskware.HackTool** [14]. Este se define como un software malicioso que puede incluir herramientas como *exploits*, *keylogger*, troyanos, entre otros, y su principal objetivo es obtener información confidencial. En este caso, el *trainer* funciona con accesos a memoria, y por ello es importante poner como excepción en el antivirus a ese ejecutable, evitando así que lo elimine sin dar aviso.

El resultado final de *trainer* se puede observar en la Figura 2.7.

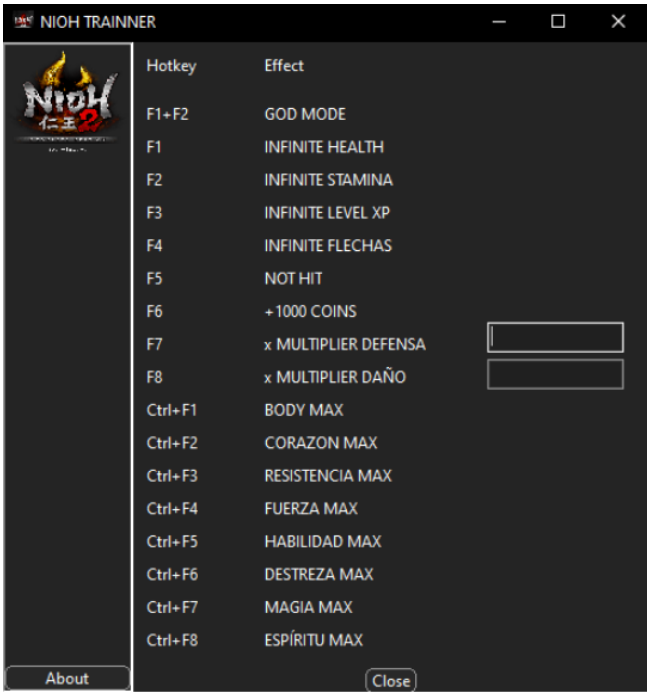


Figura 2.7: Interfaz del trainer desarrollado para Nioh

Capítulo 3

Inyecciones avanzadas

3.1. Puesta en marcha

Hasta el momento en este trabajo se han desarrollado los *scripts* apuntando a direcciones de memoria específicas. Sin embargo, este enfoque puede causar problemas en futuras actualizaciones de los juegos ya que las direcciones de memoria pueden variar, inutilizando así los *scripts* creados. Para abordar este problema, existen varias plantillas de inyección avanzadas, tales como la inyección de API que llama a código personalizado, la reubicación de código que mueve secciones de código en la memoria, la inyección Array of Bytes (AOB) que busca patrones únicos de bytes en la memoria, y por último la inyección completa que genera un proceso desde 0 y carga código personalizado en él. De todas ellas, la inyección AOB es la más recomendable, ya que se pasa de hacer inyecciones en direcciones específicas a bloques de bytes concretos, reduciendo así las posibilidades de quedar inutilizada en futuras actualizaciones. Esto no significa que las inyecciones anteriores ya no sirvan, puesto que únicamente se tendría que volver a buscar donde se ejecutaba la inyección y mover el *script*, algo que puede ser tedioso, puesto que seguramente tenga que hacerse para cada actualización.

Se realizaron varias búsquedas de posibles juegos para realizar la inyección [15], destacando aquellos juegos actuales o que sigan recibiendo actualizaciones. Algunos de los juegos probados fueron Civilization, Monster Hunter World y Nioh2, pero no se encontró mucha variedad para realizar las inyecciones. Por lo tanto, se decidió utilizar el juego Stardey Valley [16], un juego de simulación de vida en una granja desarrollado por ConcernedApe y publicado por Chucklefish. En este juego, el jugador controla a un personaje que decide abandonar su trabajo en la ciudad para heredar una granja en ruinas en el valle. Hay muchas posibilidades de inyecciones, como la modificación de estadísticas básicas, teletransporte, ratios de probabilidad en los objetos, entre otros, que se analizan en las siguientes secciones.

3.2. Código polimórfico

En el proceso de desarrollo se comenzó aplicando la nueva metodología, en la cual se requiere generar patrones de bytes únicos para evitar conflictos con otras inyecciones. Para este propósito, se implementó la siguiente estrategia:

```
1 aobscan(timeNow,48 B8 10 50 ?? ?? ?? 7F 00 00 83 00 0A)
2 7FF887EADEF4: 74 0D                          - je 7FF887EADF03
3 ; ---------- Inicio de inyección ----------
```

```
7FF887EADEF6: 48 B8 10 50 F6 86 F8 7F 00 00  - mov rax,00007FF886F65010
; ---------- Finalización de la inyección ----------
7FF887EADF00: 83 00 0A                       - add dword ptr [rax],0A
```

Se puede observar cómo se añaden caracteres que permiten generar el patrón, ya que en este caso la inyección debe situarse en un código polimórfico. La lógica a seguir para generar estos patrones depende de los tipos de instrucciones. No es lo mismo realizar una inyección en un **mov rax,r12** que en un **call 00007FF886F65010**, puesto que al deshabilitar la inyección se ensamblaría otra que podría llegar a corromper la memoria del juego. Por ello, cuanto más polimórfico sea la zona de código, mayor robustez deberá tener el patrón para generarlo.

Para garantizar que las instrucciones sean resistentes a los cambios, se recomienda:

- Analizar la estructura de versiones anteriores para destacar cuáles son las instrucciones que tienen mayor permanencia en el código, no tanto las instrucciones, sino las regiones de código que menos suelen modificarse.
- Utilizar inyecciones en regiones donde el código no sea polimórfico y que no sean instrucciones que accedan a direcciones mediantes módulos o directamente (**call 23423FA, mov edi,[23423FA]**). La mayoría de instrucciones que tiene menor probabilidad de cambio utilizan desplazamientos, no acceden a direcciones directamente, utilizan operaciones básicas y no incluyen cálculos en las instrucciones.

3.3. Limitaciones externas

Una vez conocidas las acciones para generar patrones únicos de búsqueda, se procede a realizar cambios en los valores de la estamina del personaje, una vez aplicados los procedimientos para poder controlar el código compartido se procede a manipular los valores. En este punto surge un nuevo problema, pues en ese caso los objetos del inventario no tienen una limitación definida, es decir, no hay un rango máximo. Esto llega a afectar a la propia interfaz del jugador (ver Figura 3.1).

Figura 3.1: Inventario básico del personaje

Por tanto, se identifican los máximos de cada objeto dentro de los registros de propósito general, y en ese caso **rsi** parece contener los valores tanto actuales como máximos.

Una vez que los desplazamientos han sido identificados, quedaría reemplazar la instrucción original por una que acceda a los máximos, en este caso, mediante la instrucción **movss xmm1,[rsi+4C]** (ver Figura 3.2).

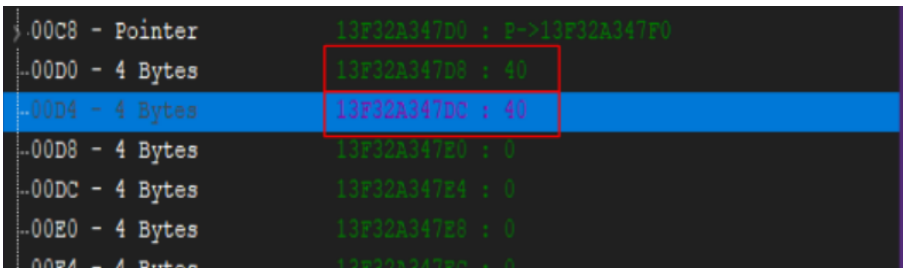


Figura 3.2: Estructura de desplazamientos dentro del puntero del agua

3.4. Inyecciones silenciosas

Al realizar una inyección que afecta al paso del tiempo dentro del juego se siguió la misma metodología, consiguiendo resultados satisfactorios. Sin embargo, al deshabilitar el *cheat* se producía un fallo que cerraba el proceso. Después de varios intentos de depuración en la zona en memoria, se descubrió que se agregaba una inyección adicional al desactivar el *cheat*. Para evitar este problema, se implementó la lectura de los bytes(**readmem**) relacionados con la instrucción y tras habilitar los *cheats* se restableció la instrucción original(**reasembly**), consiguiendo pasar silenciosamente por la ejecución del proceso.

Una vez conseguido el escaneo de memoria, se procedió a modificar el puntero que almacenaba el valor del tiempo actual. Sin embargo, se descubrió que al cambiar entre pantallas de carga, la hora se modificaba a las 02:00 a.m., lo que hacía que la partida finalizara automáticamente. Se realizó una búsqueda para ver si Stardey Valley disponía de algún tipo de *anticheat*, pero no se encontró nada. Para resolver este problema, se ejecutó una inyección que únicamente almacenaba en una variable los punteros de tiempo, sin modificar los valores. De esta manera, cuando se hace la lectura, esta queda registrada como símbolo y desde la Cheat Table se puede modificar sin generar ningún conflicto.

Tras varias pruebas se descubrió que dependiendo del momento en el que se active el *cheat*, el escáner AOB podía o no funcionar debido a que si no se ha utilizado la instrucción previamente, esta no ha compilado en memoria. Por ello, antes de utilizar cualquiera de los *scripts* es necesario previamente forzar el compilado de las instrucciones.

El único problema encontrado es debido a que el juego es un ejecutable portable, por tanto, al realizar el escaneo AOB hay que escanear toda la memoria del ordenador en busca del patrón indicado. Dependiendo de la cantidad de procesos que se estén ejecutando en el sistema, la búsqueda tardará más o menos tiempo, llegando a afectar en el rendimiento de las inyecciones y la experiencia de usuario. En algunas ocasiones este tiempo puede ser considerablemente largo, lo que implica un inconveniente en términos de eficiencia y fluidez del juego.

3.5. Inyecciones en .NET

Para resolver la problemática planteada, se determinó que el motor del juego correspondía a XNA Framework, una biblioteca de Microsoft para crear juegos en diferentes plataformas. El lenguaje de programación utilizado es C# y hace uso del *framework* .NET. Es posible identificar las llamadas dentro de la lógica del juego gracias a la capacidad de CE para relacionar las funciones .NET con las instrucciones a nivel ensamblador.

Para visualizar las funciones se optó por hacer uso de dnSpy [17], el cual permite

la descompilación de las *Dynamic Link Libraries* (DLL) del juego para acceder a las funciones .NET y visualizarlas en lenguaje C#. Esta herramienta resulta de gran utilidad al reducir los tiempos de depuración y facilitar la compresión de la lógica de las funciones. El proceso de resolución se inició con la decompilación de las DLL del Stardey Valley mediante el uso de dnSpy (ver Figura 3.3).

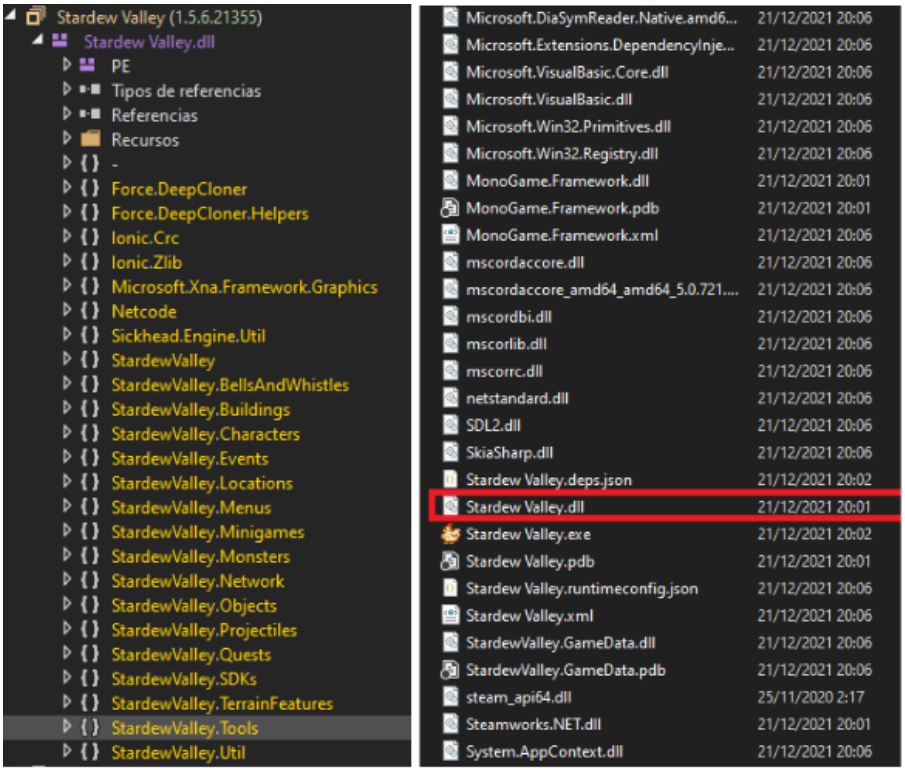


Figura 3.3: Vista de las DLL desde la interfaz de dnSpy

En la investigación llevada a cabo se evidenció la presencia de numerosos valores, estadísticas, características y utilidades dentro del juego que pueden ser accedidos. Por esta razón, se realizó una búsqueda de la función encargada de llevar a cabo el desplazamiento del personaje, identificada como **getMovementSpeed**, y ubicada dentro de la clase **farmer**, la cual representa al jugador dentro de la partida. Cabe destacar que dicha función realiza varias comprobaciones para verificar si se han aplicado ciertas características que modifiquen la velocidad, pero lo más importante es que al finalizar la función retorna un tipo flotante. Conocer el tipado de la variable devuelta por el servidor puede ayudar a realizar búsquedas en CE de manera más eficientes, sin necesidad de probar con diferentes tipos de datos.

Es importante mencionar que al instalar .NET en el equipo, CE es capaz de identificar las direcciones de las funciones con los nombres reales, lo que permite buscar la transformación de C# a ensamblador directamente desde el programa. Una vez encontrada la función en cuestión, se procede a depurar la zona y, tras varias iteraciones revisando los registros de tipo flotante **xmmY**, se puede observar que la velocidad actual y la velocidad modificada, tras pasar por la función, se almacenan en **xmm0**. De esta manera se puede modificar el valor de este registro al realizar la llamada y, posteriormente, utilizar **ret** para devolver el valor flotante esperado, eliminando así todas las condiciones que accedían a la estructura del jugador.

Gracias a este método ya no es necesario realizar un escaneo de toda la memoria, sino que se puede acceder directamente la función .NET haciendo que la inyección sea casi

inmediata. Además, estas inyecciones, que ya no se basan en identificación de patrones de bytes, se salen del esquema planeado, lo que les confiere una mayor resistencia a actualizaciones o parches que reciba en juego.

3.6. Compilación de inyecciones

Uno de los problemas mencionados previamente es que las instrucciones solo se compilan en memoria cuando el jugador utiliza una herramienta específica en el juego. Sin embargo, con las funciones .NET, este problema se agrava debido a que el momento que CE se conecta con el proceso es determinante para la identificación de los nombres de las funciones. Esto se debe al Common Language Runtime (CLR), que recopila los códigos de varios lenguajes (Visual Basic, C#) y utiliza un compilador Just In Time, haciendo que se compilen las instrucciones en tiempo de ejecución. Sin embargo, CLR no puede reiniciar los símbolos .NET, lo que dificulta aún más la tarea de identificar las funciones.

Después de investigar esta problemática, se descubrió que se puede implementar en las inyecciones la función **reinitializeDotNetSymbolhandler**. Esta función de Lua permite inicializar el manejador de símbolos .NET, lo cual es necesario después de actualizar las DLL del juego debido al uso de una herramienta no utilizada anteriormente en el juego u otras funcionalidades. La función de búsqueda hace empleo de estos símbolos, por lo que si no se encuentran, puede deberse a que no se han usado las herramientas necesarias previamente o porque no se han reinicializado la lista de símbolos .NET correctamente.

Para entender mejor el funcionamiento de lo explicado, se puede observar la Figura 3.4, donde se ve el momento exacto de aplicación para cada una de las acciones mencionadas.

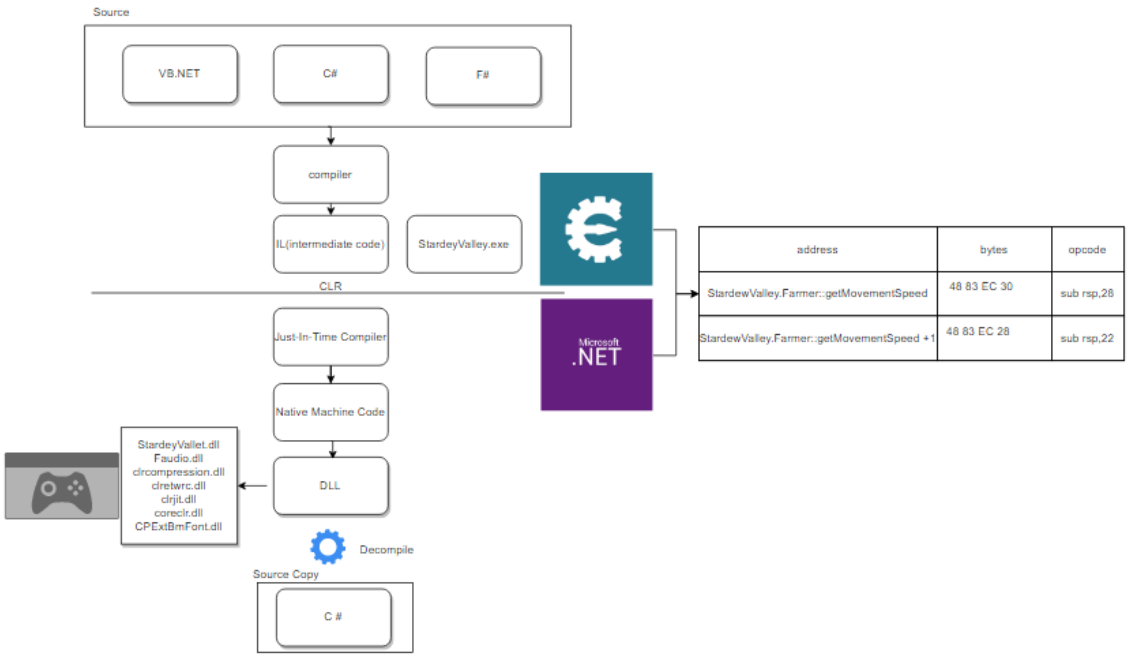


Figura 3.4: Compilación de las inyecciones

3.7. Estructura del jugador

Después de haber logrado modificar la velocidad del jugador y haber verificado que en el registro de propósito general **rcx** se almacena un puntero a la estructura del jugador (ver Figura 3.5), se procedió a buscar en dnSpy una función que únicamente retorne dicho puntero y que tenga alta probabilidad de permanecer sin cambios. Se encontró la función **get_Player**, que retorna el puntero accediendo al registro **rax**. Por esta razón, se declaró una variable que almacene dicho puntero y, desde la Cheat Table, depurar toda la estructura, agregando al *script* solo aquellos valores que tuvieran sentido y no afectaran negativamente al proceso.

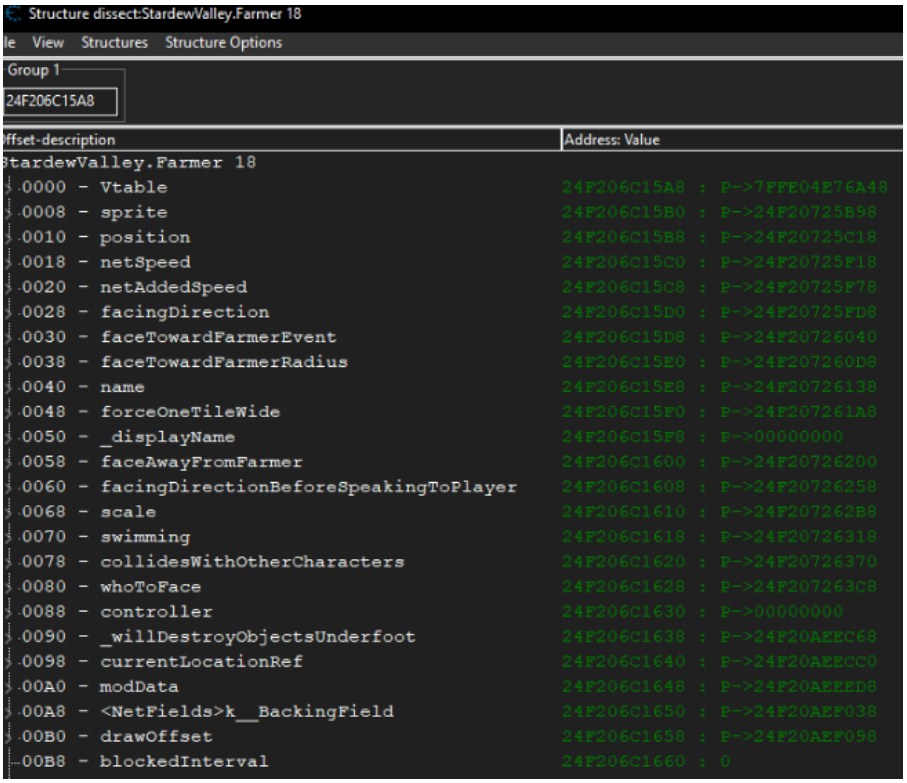


Figura 3.5: Estructura básica del jugador

Además de la funcionalidad de .NET para asignar nombres a las estructuras, se pueden agregar varios desplazamientos a la Cheat Table para que el usuario pueda modificar los valores después de haber sido depurados previamente, asegurando que el valor modificado no corrompa el proceso en ejecución. Adicionalmente, para aumentar la resistencia a las actualizaciones por parte de los desarrolladores, se pueden vincular varias funciones que tengan acceso a los registros, de modo que si se modifica una función y ya no se accede a esa estructura a través de ese registro, se pueda utilizar una función que si tenga acceso a las estructuras por otra vía. Cabe destacar el desplazamiento **farmer_ptr ->24F206C15A8, [24F206C15A8 + 10] ->24F20725C18 , [24F20725C18 + 8] ->24F20725CA8, 24F20725CA8 + 4C Float**, que hace referencia a la posición actual del jugador. Si bien se pueden modificar los valores de los ejes x e y, solo se pueden considerar los valores dentro del rango de la pantalla.

3.8. Relación entre .NET y ensamblador

Una de las actividades que resulta especialmente interesantes en términos de modificación es la actividad de pesca. Para llevar a cabo esta tarea, se puede hacer uso de la función **StardewValley.Tools.FishingRod::doPullFishFromWater**. Al examinar el código .NET, se puede ver que la función recibe varios parámetros, los cuales almacenan opciones relacionadas con los menús de pesca, tales como la captura doble, el tipo de pez, la calidad y el tamaño (ver Figura 3.6).

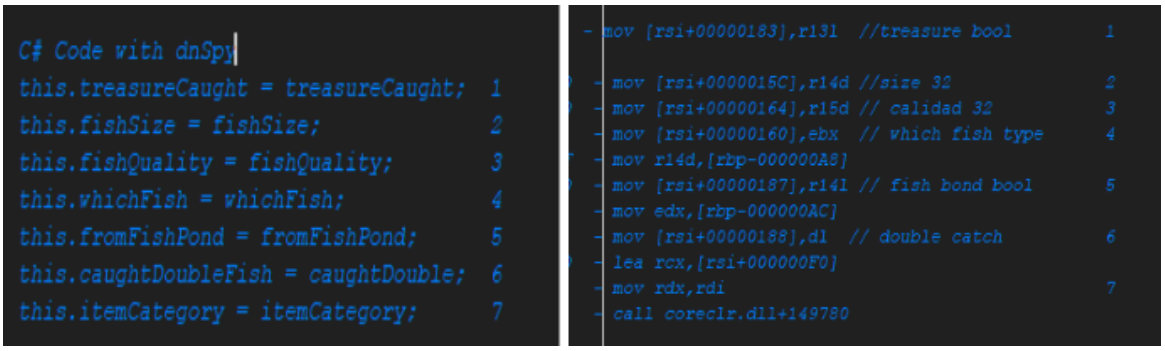


Figura 3.6: Comparativa de código C# y ensamblador en actividad de pesca

Se puede observar en la Figura 3.6 que existe un orden en el código en C# en comparación al código ensamblador. Al acceder al registro **r13d**, exactamente a los 8 bits inferiores, se puede observar que se almacena un valor *booleano*, como resultado se puede modificar estos registros por los valores indicados por el usuario y que sigue con el flujo normal de la función. Destacar que la variable **whichfish** (32 bytes) almacena el identificador del pez, y después de realizar una búsqueda se consiguió sacar una lista de todos los identificadores del juego. A partir de ahora, se puede sacar cualquier objeto del juego sin necesidad de estar en el rango esperado para los peces, lo que rompe por completo lo esperado en las funcionalidades del juego. También se podría utilizar dicha lista para alguna de las otras inyecciones anteriores.

Además de la modificación de los parámetros de pesca, también se ha accedido a la función **StardewValley.Tools.FishingRod::calculateTimeUntilFishingBite**. Al modificar el registro flotante **xmm0** a 0, se puede omitir el tiempo necesario para pescar, ya que esta función, dependiendo de las características del entorno, devuelve un flotante en un rango. Al combinar ambos *cheats*, se ha cubierto por completo la creación de un mecanismo para abusar la mecánica de pesca en el juego.

3.9. Accesos indebidos a memoria

Una de las actividades fundamentales en el juego es la agricultura. Con el objetivo de modificar los parámetros que definen la calidad de la semilla se encontró la función **StardewValley.Crop::hitWithHoe**, que permite desglosar estos parámetros mediante punteros (ver Figura 3.7). Destacan los parámetros: fase actual, fase de muestreo y la opción de replantar. Al golpear una semilla con cualquier herramienta, se ejecuta esta función, lo que permite determinar los parámetros que son necesarios para conseguir una modificación instantánea del cultivo plantado.

La función que se modifica es la que se ejecuta al plantar, **HoeDirt::plant**. Para ello, se realizan los desplazamientos en memoria utilizando los desplazamientos ya conocidos

de funciones anteriores, siempre verificando que no se realice un acceso ilegal a memoria. En caso de ser así, se vuelve al flujo normal de la función. Una vez se accede a la región específica de memoria, se procede a mover los valores del desplazamiento máximos. De esta manera se puede obtener automáticamente el cultivo luego de haberlo plantado (ver Figura 3.7).

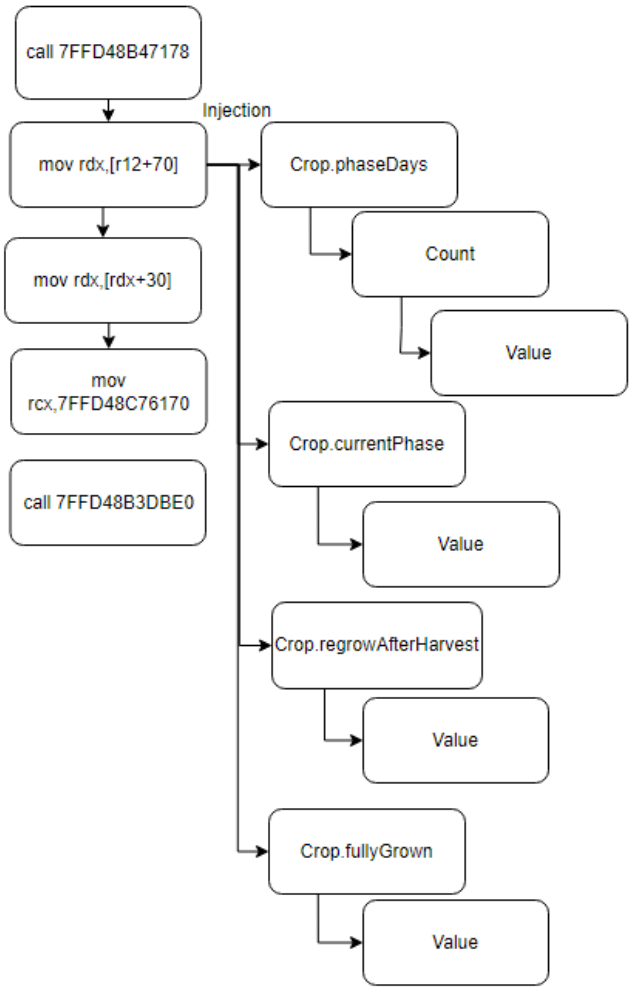


Figura 3.7: Estructura de la actividad de pesca

Como puede observarse en la Figura 3.7, se necesita acceder a los desplazamientos del objeto modificándolos en el instante de tiempo que estén disponibles.

3.10. Variedad de inyecciones

Una de las inyecciones más difíciles fue la actividad de fabricación sin requerimientos. El primer acercamiento consistió en identificar las funciones dentro de la clase **CraftingRecipe** que accedían al inventario del jugador para verificar la lista de requisitos. Se identificaron dos funciones **hasItemInInventory** y **consumeIngredients**, tales que ambas acceden al puntero de inventario del jugador y a la estructura **_count**, que modifica la lista de requisitos.

Sin embargo, al eliminar la llamada a ambas funciones y eliminar la lógica dentro de ellas, solo se logró que no se gastarán los ingredientes dentro del inventario, pero no se eliminan los requisitos necesarios para construir el objeto.

```
StardewValley.Farmer::hasItemInInventory:
  ret
StardewValley.CraftingRecipe::consumeIngredients:
  ret
```

Por lo tanto, se realizó una búsqueda adicional para acceder directamente al inventario de las recetas, lo que llevo al descubrimiento de la función **getContainerContents**. Dicha función se ejecuta constantemente mientras esté abierta la página de fabricación y tiene un parámetro dentro de la estructura **hoverRecipe** que solo tiene valor cuando el puntero del ratón está encima de algún objeto de la página, mostrando gracias a esta utilidad una lista de los requisitos necesarios para la fabricación del objeto.

Gracias a ese parámetro se puede acceder al valor **_count** mediante un puntero de 4 niveles a la lista de ingredientes. Al establecer este valor en 0, se eliminaron los requisitos necesarios para fabricar cualquier objeto. Sin embargo, es importante tener en cuenta que ese puntero solo es accesible mientras el ratón este encima del objeto, por lo que se debe probar el acceso ilegal a memoria constantemente, esperando a que el usuario pase por encima de alguno de los objetos. Además, se encontró otro puntero a ese mismo nivel que contiene el identificador del objetivo, lo que permitió utilizar nuevamente la lista que se usó para la inyección en la actividad de pesca. De esta manera se tendrían fabricaciones ilimitadas y sin requisitos, además de poder obtener cualquier objeto interno del juego (ver Figura 3.8).

Figura 3.8: Inventario de fabricación

3.11. Comandos por consola

Uno de los métodos más interesantes fue **textBoxEnter** que recibe una cadena y en caso de utilizar el carácter de barra invertida, llama a una función que es capaz de habilitar los comandos dentro del juego. Para poder introducirlo es necesario tener habilitado el **debugMode**, variable que probablemente hayan empleado los desarrolladores para agilizar los testeos de los códigos desarrollados. Para habilitar este modo, se debe acceder a la función **StardewValley.Menus.ChatBox::runCommand** y almacenar el registro que contiene la estructura con la lógica.

A continuación, se debe desplazar hasta la variable de tipo byte que corresponde a **debugMode** en el desplazamiento **5B**. Al establecerla en 1, se puede utilizar cualquier comando creado por los desarrolladores. Es interesante observar en esta función como oculta los detalles de comando real para evitar que los usuarios puedan manipularlo directamente. Esto puede ser útil para evitar que los usuarios usen trampas dentro del juego o realicen acciones no autorizadas.

```
internal static uint ComputeStringHash(string s)
  {
```

```
    uint num;
    if (s != null)
    {
  num = 2166136261U;
  for (int i = 0; i < s.Length; i++)
  {
    num = ((uint)s[i] ^ num) * 16777619U;
  }
    }
    return num;
}
```

La función emplea el algoritmo FNV-1a (Fowler–Noll–Vo) [18] para calcular el *hash* de la cadena, que es un método comúnmente utilizado para generar identificadores únicos para objetos y datos. El proceso del cálculo del *hash* se realiza mediante la iteración de los caracteres de la cadena y la realización de la operación XOR, además de la multiplicación con un número constante primo.

3.12. Punteros dinámicos

Otra de las actividades principales dentro del proceso es el acceso a las cuevas para obtener recursos. Inicialmente, se buscó el acceso a un puntero que contuviera información de la cueva en tiempo real, es decir, que se ejecutara en cada instante de tiempo. Sin embargo, se identificó una función llamada **Locations.MineShaft::isDarkArea**, la cual accede a dicha estructura por cada cambio entre los diferentes niveles de la cueva.

El usuario tiene la capacidad de modificar los niveles dentro de la mina, la probabilidad de aparición de ítems y cualquier otro aspecto relacionado con la misma. Esta función actualiza los parámetros al cambiar de nivel, lo que permite modificar dinámicamente los valores de la estructura. Además, es posible activar algunas inyecciones mencionadas anteriormente para desplazarse dentro de la mina utilizando el puntero del jugador 3.5.

3.13. Comunicación entre cliente y servidor

En esta etapa se ha evaluado la diversidad de alternativas disponibles en las inyecciones y las distintas estrategias para abordar los problemas. En consecuencia, se procedió a realizar pruebas con la versión actual del *trainer* en comparación a otras versiones. La versión actual de Stardey Valley, en el momento de la redacción de este documento, es la 1.5.6.22018 y los *cheats* se han desarrollado para la versión 1.5.6.21355. Se puede generalizar las versiones como X.X.X, siendo los cambios a la izquierda más relevantes y las actualizaciones a la derecha menores y se enfocan en corregir errores de menor importancia. En este caso, ambas versiones se diferencian por cambios menores y, por lo tanto, no deberían afectar a los *cheats* desarrollados.

Se han realizado pruebas en la versión actual del juego del uso de los *cheats*, tanto los que emplean la técnica de inyección AOB como .NET, y se ha verificado que funcionan correctamente. También se han llevado a cabo pruebas en partidas online con varios jugadores, en las que se ha confirmado que los *cheats* funcionan correctamente siempre y cuando el creador de la partida actúe como host y los demás jugadores validen sus datos realizando conexiones con este.

Para verificar esta información, se ha iniciado una partida multijugador dentro del

mismo equipo. Uno hará las operaciones del servidor y se comportará como *host* de la partida, mientras que el otro proceso de Stardey Valley hará la función de cliente que se conectara mediante la interfaz de *loopback*. Para analizar correctamente la comunicación dentro de esta arquitectura se hará uso de Wireshark [19], que permitirá capturar y analizar el tráfico de red en tiempo real. Esto permitirá examinar los paquetes enviados entre cliente y servidor, incluyendo cualquier posible interacción con los *cheats* (ver Figura 3.9).

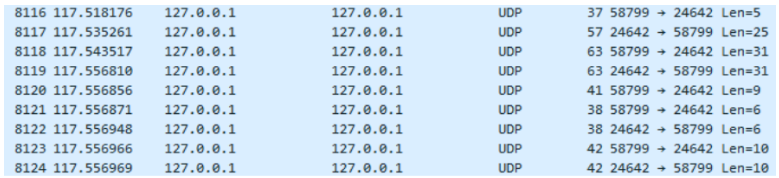


Figura 3.9: Captura de paquetes UDP

En la Figura 3.9 se pueden observar los paquetes de datos utilizados para llevar a cabo la conexión entre los procesos, los cuales indican que se está utilizando el protocolo UDP. Esta elección es inusual, ya que lo habitual en estos casos es emplear primero TCP para garantizar una conexión fiable, y luego UDP para obtener en la mayoría de casos una mayor velocidad de comunicación.

Con el fin de investigar este comportamiento, se revisaron las DLL del proceso en busca de la librería utilizada para establecer las conexiones. Se encontró que en la sección **StardeyValley.Network.GameServer** se utilizaba la librería Lidgren.Network [20]. Esta biblioteca escrita en C#, utiliza el protocolo UDP y proporciona características como predicción y corrección de posición. Además, es compatible con XNA, el motor en el que se basa Stardey Valley. Con esta información, se puede confirmar que estos procesos utilizan exclusivamente el protocolo UDP. Esto es algo realmente interesante ya que permite ver la comunicación entre ambos procesos debido a que no se realiza ningún tipo de cifrado de datos entre cliente y servidor (ver Figura 3.10).

```
private void sendVersionInfo(NetIncomingMessage message)
{
   NetOutgoingMessage response = this.server.CreateMessage();
   response.Write("1.5.5");
   response.Write("StardewValley");
   this.server.SendDiscoveryResponse(response, message.SenderEndPoint);
   if (this.bandwidthLogger != null)
   {
        this.bandwidthLogger.RecordBytesUp((long)response.LengthBytes);
   }
}
```

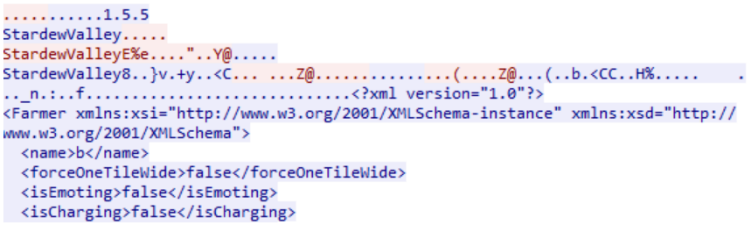


Figura 3.10: Comunicación entre cliente y servidor

En la Figura 3.10 se puede observar en rojo el inicio de la comunicación por parte del cliente y las respuestas por parte del servidor. Una de las acciones iniciales es la verificación de las versiones del proceso entre ambas entidades, debido a que si tienen diferente versión podría llegar a afectar en la jugabilidad [21]. Además, el servidor, tras verificar la conexión, envía al cliente la partida guardada para que este pueda continuar por donde lo había dejado o, en caso de ser la primera conexión, guardar esos datos en local (ver Figura 3.11).

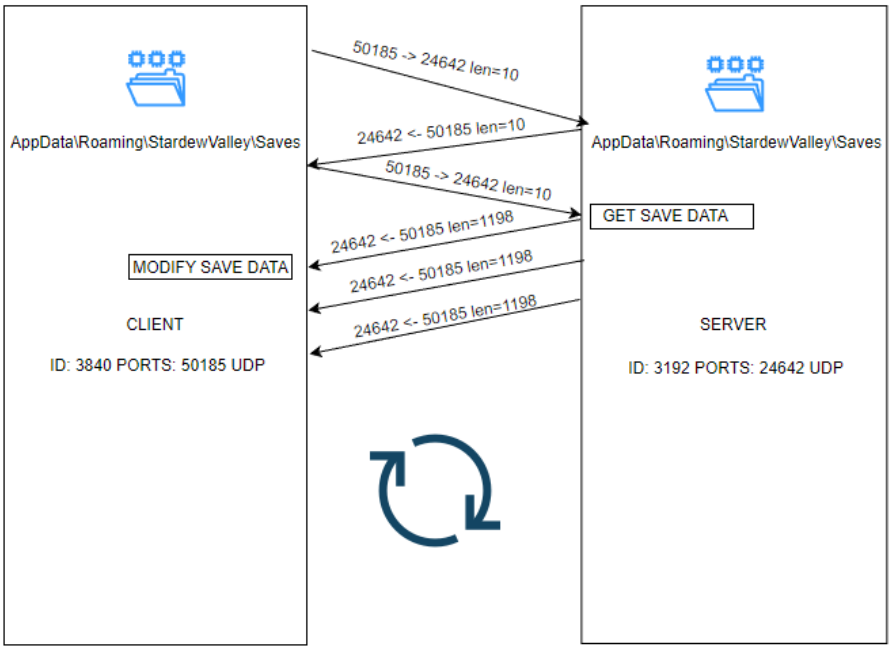


Figura 3.11: Gestión de partidas guardadas en Stardey Valley

Debido a las circunstancias se procede a obtener una versión más antigua del juego (1.3.37), en la cual los *cheats* continúan funcionando correctamente tanto mediante AOB como por direcciones. En caso de haber encontrado algún DLL que modifique considerablemente el juego, se tendría que adaptar el *cheat* correspondiente. Sin embargo, dado que el juego no ha recibido más actualizaciones, se puede concluir que las técnicas de inyecciones utilizadas y la variedad de conexiones entre las funciones y los objetivos del *cheat* han permitido que este funcione en versiones anteriores y posteriores. En caso de haber encontrado algún problema, es posible que se deba a las inyecciones basadas en direcciones, que tienden a ser variables. No obstante, en caso de fallar, bastaría con revisar el código y actualizar la zona de inyección.

Además de verificar el correcto funcionamiento, resulta interesante analizar como este afecta una partida compartida entre varios jugadores. Después de varias pruebas realizando inyecciones tanto en cliente como servidor, se llegó a la siguiente conclusión: las inyecciones que afecta al propio jugador tienen efecto en el servidor, lo que significa que el personaje del cliente puede cambiar de posición, velocidad, parámetros de jugabilidad, relaciones con los ciudadanos e ítems del inventario. Sin embargo, algunas características como el cambio de hora no se ven reflejadas en el servidor. Por el contrario, si es el servidor es quien ejecuta la modificación, esta se refleja automáticamente en el cliente debido a la replicación de datos.

Una de las inyecciones más interesante es la del dinero, ya que este es compartido entre todos los jugadores. Si se modifica el dinero en el cliente y se realiza una compra en el servidor, se utilizará el dinero del servidor. Si, por el contrario, se modifica el dinero

en el cliente y se realiza la compra en el cliente, el dinero se actualizará en el cliente. Es decir, el desarrollador supone que el dinero en el cliente y en el servidor es siempre igual, por lo que actualiza el valor de ambos con la cantidad que se ha utilizado para realizar la compra sin importar que parte realizó la compra.

Capítulo 4

Evasión de sistemas anticheats

4.1. Mecanismos de anticheats

En este capítulo se aplica la técnica de inyección en un juego con *anticheat*, con el objetivo de evitar su detección. Se utiliza el juego Rising Hell [22], el cual está desarrollado en Unity, lo que permite descompilar las DLL con relativa facilidad. En este juego, el jugador encarna un personaje 2D que se encuentra en el infierno y tiene que luchar con los demonios para escapar. El juego ofrece diferentes personajes y habilidades para desbloquear mientras te abres camino a través de un mundo generado al azar lleno de criaturas peligrosas, jefes y trampas. Se aplican aquí las técnicas aprendidas en capítulos anteriores para modificar el comportamiento del juego.

Al comenzar con una modificación de valor desde el propio CE, el juego lanza un mensaje en pantalla advirtiendo de que se está intentando modificar la memoria. También sucede lo mismo realizando una inyección AOB (ver Figura 4.1).

Figura 4.1: Mensaje de cheat detectado en Rising Hell

Lo primero que se puede observar en la carpeta del juego **/RHBuild.824164/Rising Hell/_Data/Managed** es que contiene varias DLL interesantes, entre ellas Anticheat.dll y Assembly-CSharp.dll. En esta última se encuentra toda la lógica de funcionamiento del juego. Tras comparar en el depurador algunas instrucciones que afectaban a la vida del personaje con la lógica de la DLL (ver Figura 4.2), aparecen algunas llamadas que no están dentro de esta. Por ello se analizaron las DLL del *anticheat*.

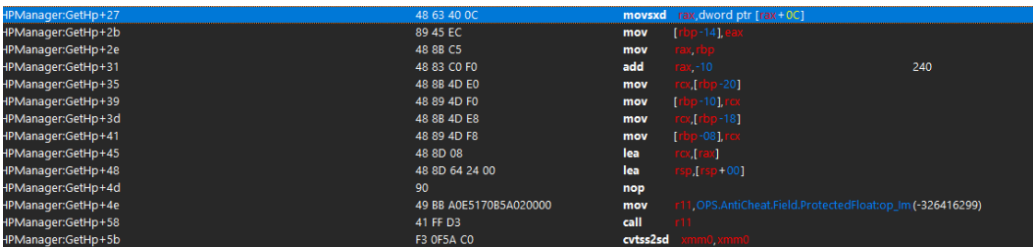


Figura 4.2: Depurador de CE

Si se observa una de las funciones que se llama implícitamente en la gestión de la vida del juego, se encuentra **OPS.AnticheatField.ProtectedFloat_Im**. El hecho de que reciba un valor flotante puede ser útil para reducir los tiempos de búsqueda. Se procede a buscar con los datos anteriores dicha función en los archivos locales del juego.

```
public ProtectedFloat(float value = 0f)
{
  this.randomSecret = (uint)ProtectedFloat.random.Next(0, int.MaxValue);
  this.securedValue.intValue = 0U;
  this.securedValue.floatValue = value;
  this.securedValue.intValue = (this.securedValue.intValue ^ this.randomSecret);
  this.manager.intValue = 0U;
  this.manager.floatValue = 0f;
  this.fakeValue = value;
}
```

En la función se pueden observar los siguientes aspectos:

- Generación de número aleatorio como un entero sin signo.
- Se establece un par con un 0 y el valor recogido por parámetro, además se realizará una operación XOR entre el número aleatorio y el 0.
- Finalmente, el valor resultante se almacena en **this.fakeValue**.

En general, esta función tiene como objetivo proteger el valor de punto flotante de ser manipulado por un atacante. La operación XOR camufla el valor real de todas las variables que estén protegidas por esta función. Además de esta, el juego dispone de varias funciones para cada tipo de dato del que hace uso (ver Tabla 4.1).

Tipo de dato	**Función**
int16	OPS.AnticheatField.ProtectedShort_Im
int32	OPS.AnticheatField.ProtectedInt_Im
int64	OPS.AnticheatField.ProtectedLong_Im
string	OPS.AnticheatField.ProtectedString_Im
uint16	OPS.AnticheatField.ProtectedUShort_Im
uint32	OPS.AnticheatField.ProtectedUInt_Im
uint64	OPS.AnticheatField.ProtectedULong_Im
float	OPS.AnticheatField.ProtectedFloat_Im
double	OPS.AnticheatField.ProtectedDouble_Im

Tabla 4.1: Funciones encargadas dentro del *cheat*

En primer lugar, se modificó la vida del personaje mediante una inyección en el código del juego. Para ello, se interceptó la función que se encarga de proteger el valor de la

vida mediante una operación de *hash*. Una vez localizada, se eliminó su contenido usando la instrucción **ret**.

Al realizar la modificación de la vida, no aparece el mensaje por pantalla y aparentemente sigue sin poder modificarse la vida del personaje. Sin embargo, al investigar más a fondo se encontró que posteriormente se ejecuta una llamada a una función de seguridad que bloquea las acciones, específicamente la función **FieldCheatDetector:OnCheatDetected**. Después de bloquear ambas se consiguió modificar los valores del jugador pasando desapercibido por el sistema.

Para seguir explorando el código y buscar vulnerabilidades, se decidió cambiar el enfoque y utilizar funciones encargadas de decrementar los valores de estadísticas básicas, siempre habilitando previamente el *cheat* inutilizando la comprobación de modificación en memoria.

Ha sido interesante gracias a la descompilación entender como un *anticheat* protege los accesos a memoria y el contenido de las variables, modificándolas y almacenándolas para comparar continuamente si el valor se ha modificado. En este caso ha sido posible descompilar las DLL y entender como funciona, pero en muchos otros casos estas vendrán cifradas y solo se podrá hacer uso del depurador para analizar en que punto se hacen las llamadas para proteger las variables.

4.2. Anticheat de Riot Vanguard

Una vez se ha logrado realizar las inyecciones y *bypass* en sistemas *anticheat* específicos, se procede a modificar la herramienta que permite crear las inyecciones y depurar la memoria. Entre alguna de las técnicas conocidas se tiene la ofuscación de código, cifrado, compresión, modificación de controladores y firmas dentro del programa.

Entre las herramientas más conocidas y utilizadas se encuentra Easy Anti-Cheat, Battle Eye y Vanguard, cada una con su propia tecnología y estrategia para detectar y prevenir *cheats* dentro de juegos en línea. Es importante mencionar que estas herramientas no son perfectas y, en ocasiones, pueden generar falsos positivos o no detectar todas las trampas, por lo que los desarrolladores continúan trabajando para mejorar su efectividad [23]. Para realizar las pruebas en este ámbito, se utilizará el juego League of Legends [24], el cual es capaz de detectar la ejecución de herramientas como CE, gracias a su sistema *anticheat* Riot Vanguard. Este sistema se inicia en segundo plano una vez ejecutado el proceso del juego para detectar y prevenir trampas, lo que se puede comparar con la instalación de un *spyware*, que analiza los archivos del equipo en busca de indicadores de trampas. Estos servicios llegan en algunas ocasiones a rozar los derechos de privacidad de sus usuarios [25], dependiendo del nivel de intrusión en el sistema.

Este *anticheat* actualmente se ejecuta en el anillo 3 haciendo uso de llamadas a API de Microsoft, lo que significa que, aunque esté en segundo plano, tendrá limitaciones debido a su capa de acceso. Además, no podrá detectar trampas que se ejecuten al iniciar el sistema operativo, como si podría hacer un proceso ejecutado en anillo 0.

4.2.1. Modificación de CE

Para evitar ser detectados por el proceso League of Legends_client al ejecutar CE, se descargó el código fuente de la última versión estable y luego se procedió a descargar Lazarus [26], un entorno de desarrollo integrado de software libre y multiplataforma

que permitirá modificar el ejecutable original de CE. Primero, se modificó el nombre del controlador en el archivo **dbk32/DBK32functioms.pas**, donde se encuentra la función, **DBK32Initialize**, que permite modificar el nombre de los controladores tanto en x64 como x32.

Para evitar problemas futuros, se recomienda utilizar nombres de controladores aleatorios para evitar posibles coincidencias con registros guardados en bases de datos como la de VirusTotal [27]. Este servicio en línea proporciona análisis de archivos y URL mediante el uso de múltiples motores de antivirus, manteniendo en una base de datos los archivos y *hash* que se han enviado para su análisis (ver Figura 4.3).

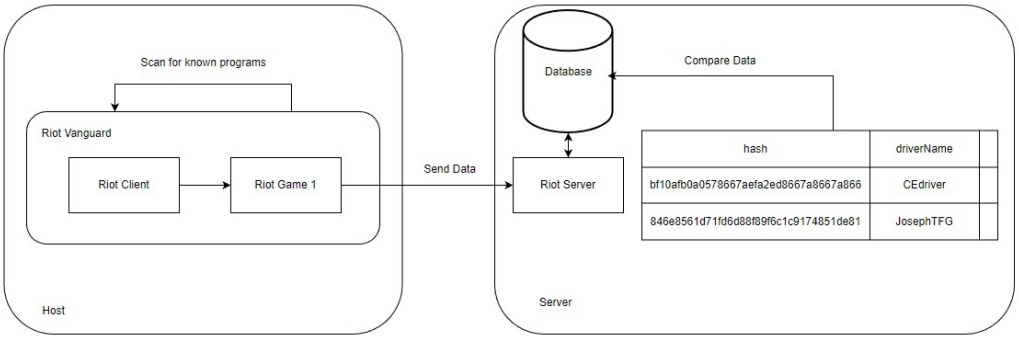


Figura 4.3: Detección de procesos conocidos en la base de datos

Luego se abre el ejecutable con Lazarus, y dentro de este se vuelve a compilar CE para comprobar que funciona correctamente. Una vez se sabe que ha compilado, se procede a modificar las cadenas que contengan el nombre del programa tanto dentro del código como en la interfaz, así como cambiar la versión y logos utilizados por la herramienta (ver Figura 4.4).

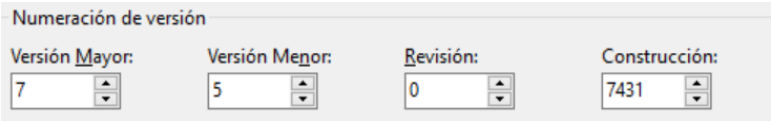


Figura 4.4: Versión actual de CE

También es importante modificar ruta *bin/JosephTFG-/(TargetCPU)-SSE4-AVX2* para que al realizar la compilación el ejecutable tenga un nombre distinto a Cheat Engine (ver Figura 4.5).

Aunque se hayan realizado estas modificaciones, es posible que haya zonas de la aplicación que conserven los nombres, lo que puede ser detectado por algunos *anticheats* capaces de identificar una secuencia de bytes o patrón de código en el ejecutable. Esta técnica es llamada *signature*, se asemeja a la inyección AOB, por ello, se eliminarán todas las firmas relacionadas con Cheat Engine y sus posibles combinaciones. Para eliminarlas se hará uso de ImHex [28], un editor hexadecimal gratuito y de código abierto. Es una herramienta útil para editar archivos binarios en situaciones en las que se necesita editar datos en bruto, como en la corrección de errores de archivos o en la edición de archivos ejecutables. En la Figura 4.6 puede observarse la interfaz del programa una vez abierto el ejecutable modificado de CE.

Para asegurarse de una correcta modificación, se comprueban todos los posibles casos, además del cambio en la codificación(UTF-8, UTF-16 y UTF-32). Al realizar pruebas con la versión actual y sus respectivas modificaciones, se confirma que no se ha conseguido

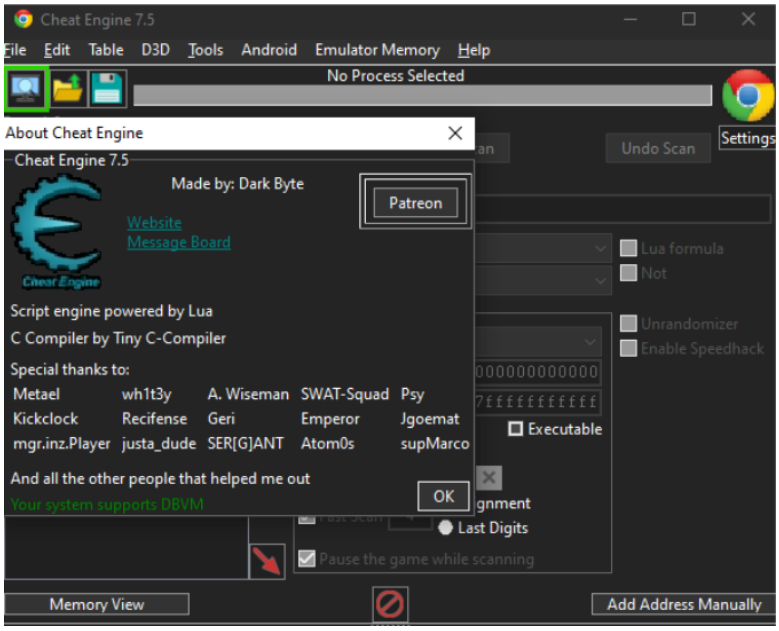


Figura 4.5: Interfaz modificada de CE

```
01040BC0  64 65 54 6F 70 2E 53 69 64 65 07 09 61 73 72 42  deTop.Side..asrB
01040BD0  6F 74 74 6F 6D 04 4C 65 66 74 02 15 06 48 65 69  ottom.Left...Hei
01040BE0  67 68 74 02 13 04 48 69 6E 74 06 E0 4E 6F 72 6D  ght...Hint.àNorm
01040BF0  61 6C 6C 79 20 4A 6F 73 65 70 20 54 46 47 6E 65  ally Josep TFGne
01040C00  77 20 77 69 6C 6C 20 64 65 6C 65 74 65 20 74 68  w will delete th
01040C10  65 20 74 72 61 63 65 20 66 69 6C 65 73 20 61 66  e trace files af
01040C20  74 65 72 20 74 68 65 79 20 68 61 76 65 20 62 65  ter they have be
01040C30  65 6E 20 70 72 6F 63 65 73 73 65 64 2E 20 42 75  en processed. Bu
```

Figura 4.6: Bloque de bytes en ImHex

saltar el *anticheat*. Esto puede ser debido a que haya firmas que no se hayan podido modificar, por lo que se utilizará la herramienta VMProtect [29]. Este es un software de protección y cifrado de código para aplicaciones de Windows. Se utiliza para proteger programas ejecutables contra ingeniería inversa, análisis de código y piratería. VMProtect emplea una variedad de técnicas para proteger los programas, incluyendo la ofuscación del código y el cifrado de datos. También es capaz de proteger programas contra ataques de *cracking* y evasión de depuración. Después de aplicar la función de mutación y modificación más segura, ya no quedará rastro de las firmas. Se puede observar en la Figura 4.7 como se vería el proceso modificado en el sistema operativo.

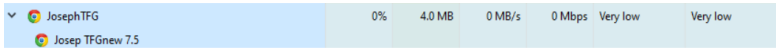


Figura 4.7: Vista del proceso desde el visor de tareas

Finalmente, se modifican los componentes del interior del ejecutable con un *script* de Lua que recorre todos los campos dentro de la interfaz y los va modificando para estar más seguros. Al enviar el ejecutable por VirusTotal, es detectado por detección de manipulación respecto al original, pero no tiene asociadas las etiquetas del ejecutable original (ver Figura 4.8).

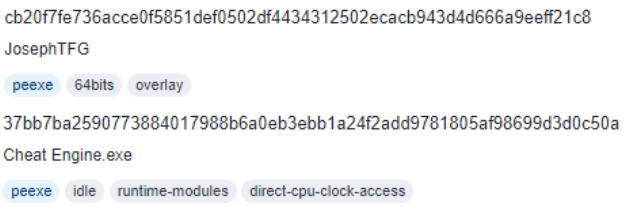


Figura 4.8: Comparación de resultado entre CE original y modificado en VirusTotal

4.2.2. Lectura de memoria y evasión

Una vez se ha logrado modificar CE para que se ejecute junto con el proceso del cliente de Riot, se pueden realizar diversas pruebas para analizar el código dentro de memoria. Sin embargo, el acceso a las DLL está cifrado, lo que significa que no se podrá analizar la lógica del juego, y solo se podrá utilizar el depurador de CE.

Aunque es importante tener en cuenta que la modificación del cliente puede llevar a la prohibición permanente de la cuenta, se debe tener mucho cuidado al realizar cualquier tipo de cambio en el juego [30]. Por lo tanto, es recomendable tomar medidas preventivas para evitar ser detectados por los sistemas de seguridad.

Entre las medidas que se pueden tomar para prevenirlos se encuentra:

- Ejecución del cliente en un entorno virtualizado. Utilizando herramienta como VMWare [31], que permiten instalar máquinas virtuales dentro del sistema operativo, pudiendo evitar detecciones basadas en componentes físicos, ya que el sistema operativo virtualizado emula *hardware* diferente al de la máquina huésped.

- Uso de una red virtual privada para prevenir prohibiciones basadas en IP (Internet Protocol). Esta técnica permite al usuario modificar su dirección IP, pudiendo editarla en cualquier momento. De esta manera es posible volver a conectarse a los servicios con la creación de otra cuenta.

- Evitar modificación de valores relacionados con la interacción con otros jugadores. Al intentar modificar el valor de la posición de un jugador o aumentar su daño es muy probable que sea detectado por el servidor debido a que este tiene la lógica y realiza operaciones para prevenir esto. Se deben seleccionar los valores que modifiquen únicamente al cliente. Esto se pudo evidenciar en el capítulo 3.

- Usar una cuenta secundaria para minimizar los riegos de perder todo el progreso y el historial de la cuenta principal.

- Utilizar diferentes patrones cuando se estén realizando pruebas de inyección, ya que puede existir patrones detección repetitivos.

Una vez tomadas las medidas anteriores se procede a anclar CE al proceso cliente de Riot. Destacar que por cada partida se ejecuta un nuevo cliente temporal que tiene la misma duración que la partida del juego, por tanto, no es lo más recomendable para CE elegir este cliente.

Para analizar las llamadas del sistema, cuando se utiliza CE se hará uso de DebugView [32], que permitirá analizar los registros de cada proceso dentro del sistema. Al realizar la conexión del CE original, no se observan llamadas sospechosas fuera del registro de cuanto tiempo el usuario pasa en la ventana del proceso. Sin embargo, al realizar el escaneo de memoria, en el cliente se detiene y envía información del equipo.

```
Sending telemetry to schema stopRCFull with: {"actionType":"stopRCFull","
    riotclientMetadata":{"appName":"Riot Client","machineId":"
    suWndnpVyUOXOYqUwMuW6s==","os":"windows","osVersion":"10 Home, x64","
    patchline":"KeystoneFoundationLiveWin","sessionId":"6a6cT76d-0a47-0349-
    b3fb-298bf1d42995","version":"64.0.2.6023401"}}
```

En el caso del CE modificado, al realizar la conexión con el cliente no ocurre nada en las llamadas. Además se pueden ver en memoria los módulos cargados correctamente (ver Figura 4.9).

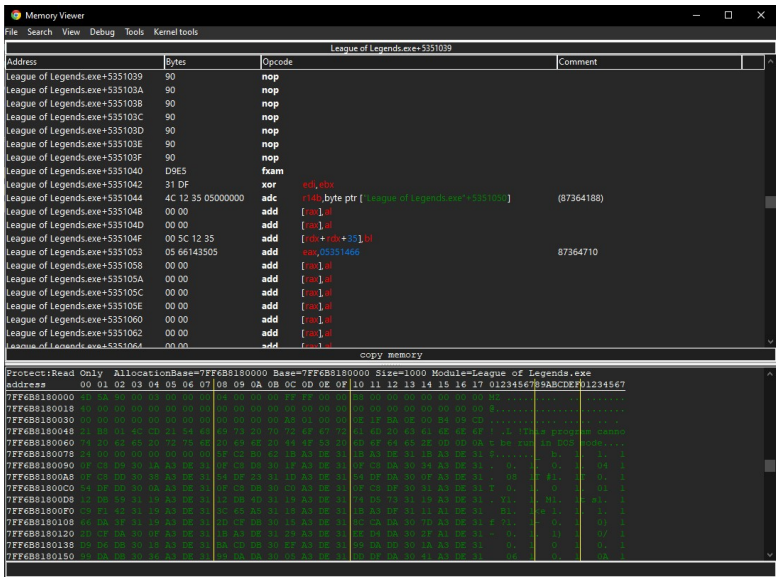


Figura 4.9: Depuración de la memoria en League of Legend

El proceso de modificación de variables e inyección en el juego ya puede ser iniciado. Sin embargo, es fundamental tener en cuenta que debido al funcionamiento del mismo, muchos de los valores que se modifiquen no tendrán efecto significativo en el juego. Esta limitación se debe a que la mayoría de cálculos necesarios para el correcto funcionamiento del juego se realizan en el servidor.

A pesar de lo mencionado, la comunidad ha creado *cheats* que permiten modificar algunos aspectos del juego, tales como la apariencia del personaje o la capacidad de lanzar ataques automáticamente a los enemigos, sin que esto afecte al lado del servidor. No obstante, se debe ser precavido con la modificación, dado que, como se ha mencionado anteriormente, aunque se utilicen todos los métodos de evasión disponibles, nunca se está cien por ciento seguro de no ser detectado.

Capítulo 5

Conclusiones y líneas futuras

En este trabajo se ha investigado el uso de técnicas de ingeniería inversa e inyección de código en juegos para modificar su comportamiento, especialmente en la creación de *cheats*. Para ello se han explorado diferentes técnicas de inyección de código, como las basadas en direcciones y patrones de bytes, que permiten encontrar regiones de memoria donde anclar las inyecciones dentro de la memoria del sistema.

Además, se ha evaluado la capacidad de los *cheats* para adaptarse a las diferentes versiones de los juegos, así como la efectividad de los sistemas *anticheats*. En este sentido, se han aplicado diversas formas de evadir la detección, como la eliminación de las firmas en el ejecutable del juego, y también se ha explorado la posibilidad de modificar los paquetes que se intercambian entre cliente y servidor para obtener ventaja en el juego. Esto ha sido posible debido al protocolo de comunicación y a la inexistencia de cifrado de paquetes entre ambos.

No obstante, en este trabajo se ha podido confirmar que la seguridad de los juegos ha evolucionado en respuesta a estas técnicas de inyección, de forma que ya se están implementando sofisticadas medidas para detectar y prevenir estás prácticas. Algunas de esas medidas incluyen la ejecución de procesos en segundo plano con privilegios en anillo 0, para monitorear las acciones en memoria y la virtualización de procesos. Cuando se ejecuta el cliente en una máquina virtual, se puede de esta forma intervenir en la memoria reduciendo la probabilidad de ser detectado. Aun así, algunos *anticheats* ya traen detectores basándose en la lectura de procesos, registros y servicios propios de una máquina virtual. También cabe mencionar una tendencia actual hacia la eliminación de clientes en local, para la ejecución remota de juegos, como por ejemplo, plataformas como GeForce Now [33].

En este contexto, la ejecución de clientes en servidores remotos puede ser una opción interesante para reducir la efectividad de los *cheats* y garantizar una experiencia de juego más justa. Sin embargo, esta opción todavía no es muy popular entre los jugadores debido al retardo que se produce en la transmisión de datos entre cliente y servidor, lo que puede afectar negativamente a la jugabilidad. Por lo tanto, todavía hay mucho por investigar y mejorar en el ámbito de la seguridad en los juegos para garantizar una experiencia justa y equitativa para todos los jugadores.

Capítulo 6

Conclusions and future works

In this work we have investigated the use of reverse engineering and code injection techniques in games to modify their behavior, especially in the creation of cheats. For this purpose, different code injection techniques have been explored, such as those based on addresses and byte patterns, which allow finding memory regions where to anchor the injections inside the computer's memory.

In addition, the ability of cheats to adapt to different game versions has been evaluated, as well as the effectiveness of anticheat systems. In this regard, ways to evade detection, such as removing signatures in the game executable, have been implemented, and the possibility of modifying the packets exchanged between client and server to gain advantage in the game has also been explored. This is possible due to the communication protocol, in addition to the non-existence of packet encryption between the two.

However, game security has evolved in response to these injection techniques, and robust measures are being implemented to detect and prevent these practices. Some of these measures include running background processes with ring 0 privileges, to monitor in-memory actions and process virtualization. To proceed, they run the client in a virtual machine, thus being able to intervene in memory and being less likely to be detected, although some anticheats already have detectors based on the reading of processes, logs services of a virtual machine. Also mention a trend towards the elimination of local clients for remote execution of games, mentioning platforms such as GeForce Now [33].

In this context, running clients on remote servers can be an interesting option to reduce the effectiveness of cheats and ensure a fairer gaming experience. However, this option is still not very popular among gamers due to the delay in data transmission between client and server, which can negatively affect gameplay. Therefore, there is still a lot of research and improvement to be done in the area of gaming security to ensure a fair and equitable experience for all players.

Capítulo 7

Presupuesto

En esta sección se realiza una propuesta del presupuesto estimado para el desarrollo del Trabajo de Fin de Grado. Como se muestra en la Tabla 7.1, se detalla el presupuesto, donde se especifican las diferentes actividades, sus horas y costes, con una estimación de por hora de 30€.

Tabla 7.1: Presupuesto

Actividad	**Horas**	**Coste(€)**
Investigación funcionamiento de la memoria en Windows	30	900
Adaptación a las posibilidades del proceso	15	450
Desarrollo de inyecciones	180	5400
Creación de la Interfaz	15	450
Bypass del anticheat	70	2100
Modificación de CE	30	900
Redacción de la memoria	40	1200
Total	380	11400

Apéndice A

Apéndice 1

A.1. Cheats de Nioh

```
[ENABLE]
; Nivel infinito del personaje.
alloc(newmem,2048,"nioh.exe"+4B07D8)
label(returnhere originalcode exit)

newmem:
; Almacena un valor de dinero muy alto.
mov rbx,#100000000000

originalcode:
mov [rsi+08],rbx
; Almacena en la direccion que guarda el dinero el valor del registro.
cmp word ptr [rdi+0C],00

exit:
jmp returnhere
; Realiza un salto a newmem, bloque que se ha reservado anteriormente.
"nioh.exe"+4B07D8:
jmp newmem
; Inutiliza 4 instrucciones debido al almacenamiento de bytes en el jmp.
nop 4
returnhere:

[DISABLE]
dealloc(newmem)
"nioh.exe"+4B07D8:
mov [rsi+08],rbx
cmp word ptr [rdi+0C],00
//Alt: db 48 89 5E 08 66 83 7F 0C 00
end
```

```
[ENABLE]
; Habilidad infinita del personaje.
alloc(newmem,2048,"nioh.exe"+4B6C91)
label(returnhere originalcode exit)

newmem:
; Nivel máximo de habilidad.
mov esi,(int)200

```

```
originalcode:
mov [rbx+000AE5C8],esi

exit:
jmp returnhere

"nioh.exe"+4B6C91:
jmp newmem
nop
returnhere:

[DISABLE]
dealloc(newmem)
"nioh.exe"+4B6C91:
mov [rbx+000AE5C8],esi
//Alt: db 89 B3 C8 E5 0A 00
```

```
[ENABLE]
; Edición de flechas en el inventario.
alloc(newmem,2048,"nioh.exe"+4A9338)
label(returnhere originalcode exit)

newmem:
;Opcode
;arrows R12 01  R13 00
;other  R12 00  R13 0A

cmp r12,0
je originalcode
; Aumenta cantidad de flechas.
mov r15w,#-2

originalcode:
; Disminuye cantidad en 1.
sub [rbp+00000218],r15w

exit:
jmp returnhere

"nioh.exe"+4A9338:
jmp newmem
nop 3
returnhere:

[DISABLE]
dealloc(newmem)
"nioh.exe"+4A9338:
sub [rbp+00000218],r15w
//Alt: db 66 44 29 BD 18 02 00 00
```

```
[ENABLE]
; Estamina infinita del personaje.
alloc(newmem,2048,"nioh.exe"+2A8A3E)
label(returnhere originalcode exit)
alloc(auxStamina,4,"nioh.exe"+2A8A3E)

auxStamina:
dd (float)9999.99
```

```
;Opcode
;RCX        PLAYER ENEMIES
;OFFSET 78   1      0
newmem:
  cmp [rcx+78],1
  jne originalcode
  ; Modificación del registro flotante.
  movss xmm2,[auxStamina]
  movss [rcx],xmm2
  comiss xmm3,xmm1
  jmp exit

originalcode:
  ; En caso de ser enemigo reducir estamina a 0.
  movss [rcx],xmm2
  mov [rcx],0
  comiss xmm3,xmm1

exit:
  jmp returnhere

"nioh.exe"+2A8A3E:
  jmp newmem
  nop 2
  returnhere:

[DISABLE]
  dealloc(*)
  "nioh.exe"+2A8A3E:
  movss [rcx],xmm2
  comiss xmm3,xmm1
  //Alt: db F3 0F 11 11 0F 2F D9
```

```
[ENABLE]
; Invencibilidad.
alloc(newmem,2048,"nioh.exe"+2A04F3)
label(returnhere originalcode exit)

newmem:
; OPCODE
; A8 PLAYER 1 ENEMIES 0
cmp [rbx+A8],1
; Salta si eres enemigo y pone vida a 0.
jne originalcode

;JUGADOR
sub eax,edi                  ; Resta a la vida el daño de ataque.
mov eax,[rbx+8]              ; Actualiza vida al máximo.
test eax,eax                 ; Comprueba  si es <= 0.
jle nioh.exe+2A051D          ; Si lo es salta a instrucción de muerte.
jmp exit

;ENEMIGOS
originalcode:
sub eax,edi
mov eax,0
test eax,eax
jle nioh.exe+2A051D
```

```

exit:
jmp returnhere

"nioh.exe"+2A04F3:
jmp newmem
nop
returnhere:

[DISABLE]
dealloc(newmem)
"nioh.exe"+2A04F3:
sub eax,edi
test eax,eax
jle nioh.exe+2A051D
//Alt: db 2B C7 85 C0 7E 24
```

```

[ENABLE]
; Inyección Completa.
alloc(newmem,2048,"nioh.exe"+2A04F3)           ; Región para inyección.
label(returnhere originalcode exit MultiplicadorDeDanoJugador ComprobarJugador
    ComprobarVidaInfinita DanoAlJugador OneHitKill ComprobarOneHitKill)  ; Etiquetas de
    salto.
alloc(dividir,4,"nioh.exe"+2A04F3)             ; Variable reductora.
alloc(multiplicar,4,"nioh.exe"+2A04F3)         ; Variable amplificadora.
alloc(sRDX,8,"nioh.exe"+2A04F3)                ; Registros para dividir
alloc(sRAX,8,"nioh.exe"+2A04F3)
alloc(sRCX,8,"nioh.exe"+2A04F3)
dividir:
  dd #1

multiplicar:
  dd #1

; Registra las variables en memoria.
registersymbol(dividir multiplicar)

; Flujos
; Jugador - Si no tiene multiplicadorDefensa - actualiza a vida máxima.
; Jugador - Si tiene multiplicadorDfensa - actualiza la vida a daño reducido(si es cero
    se incrementa en 1).
; Enemigo - Si multiplicador daño es 0 - vida enemigo = 0.
; Enemigo - Si multiplicador daño != 0 - vida enemigo = vida enemigo - daño amplificado.

newmem:

    ComprobarJugador:
        ; INICIO DE INYECCION.
        cmp [rbx+A8],1              ; Comprobar en rbx si el offset A8 es 1.
        jne ComprobarOneHitKill     ; Salta a la dirección original si no es un jugador.

    ComprobarVidaInfinita:
        cmp [dividir],0             ; Comprobar si el valor de dividir es 0.
        jne DanoAlJugador           ; Si no es 0 salta a multiplicador de defensa.

    JugadorVidaInfinita:
        mov eax,[rbx+8]             ; Eax = Vida actual.
```

```
        mov [rbx+10],eax             ; Vida actual = Vida maxima.
        jmp exit

    DañoAlJugador:
      ; JUGADOR.
      mov [sRDX],rdx                 ; Guarda el valor actual del registro rdx.
      mov [sRAX],rax
      mov [sRCX],rcx

      ;La variable de daño es de 4byte([e]di 4byte) por tanto trabaja en 32bit(mode
    hardcode).
      ;Utilizar(edx:eax/ecx) (eax:cociente,edx:resto).
      mov edx,#0                     ; 0:eax/ecx.
      mov eax,edi                    ; 0:DAMAGE/ecx.
      mov ecx,[dividir]              ; 0:DAMAGE/5.

      idiv ecx                       ; Ejecuta la división de eax por ecx, el resultado
    queda en eax(cociente) y edx(resto).

      mov edi,eax                    ; Actualiza el daño de ataque con el cociente de la
    división.

      mov rdx,[sRDX]                 ; Restaura los valores de los registros.
      mov rax,[sRAX]
      mov rcx,[sRCX]

    ComprobarDaño0:
      cmp edi,0                   ; Comprobar si el daño es 0.
      jne originalcode            ; Si no es 0 salta a la dirección original.
      inc edi                     ;Si es 0 incrementa el daño en 1(ya que si aumenta mucho
     la defensa del jugador, este nunca recibirá daño).

    ComprobarOneHitKill:
      cmp [multiplicar],0             ; Comprobar si el multiplicador es 0.
      jne MultiplicadorDeDañoJugador  ; Si no es 0 salta a la dirección original.

    OneHitKill:
      mov eax,0

    MultiplicadorDeDañoJugador:
      imul edi,[multiplicar]         ; Almacena el daño del ataque multiplicado por el
    valor de multiplicar en edi.

; ENEMIGO.
originalcode:
    sub eax,edi
    test eax,eax
    jle nioh.exe+2A051D

exit:
    jmp returnhere

"nioh.exe"+2A04F3:
    jmp newmem
    nop

returnhere:
```

```
92
93  [DISABLE]
94      dealloc(*)                  ; Liberar bloques reservados.
95      unregistersymbol(*)         ; Liberar los símbolo almacenados.
96
97      "nioh.exe"+2A04F3:
98      sub eax,edi
99      test eax,eax
100     jle nioh.exe+2A051D
101     //Alt: db 2B C7 85 C0 7E 24
```

A.2. Cheats de Stardey Valley

A.2.1. Utilizando patrón AOB

```
1   { ; Game   : Stardew Valley.exe
2     ; Fecha   : 2023-03-21
3     ; Este script realiza una escucha sileciosa del puntero de tiempo.
4   }
5
6   [ENABLE]
7   ; Patrón único en memoria.
8   aobscan(timeNow,48 B8 10 50 ?? ?? ?? 7F 00 00 83 00 0A)
9   alloc(newmem,$1000,timeNow)
10  ;label(code return time_bytes)
11  ;registersymbol(time_bytes)
12
13  registersymbol(time_ptr)
14  alloc(time_ptr,10)
15
16  ;newmem:
17    ;time_ptr:
18      ;readMem(timeNow+2,10)
19      ;jmp return
20    ;time_bytes:
21      ;readMem(timeNow,10)
22      ;jmp return
23
24  ;code:
25    ;reassemble(timeNow)
26    ;jmp time_ptr
27
28  time_ptr:
29    ; Almacena bytes para registrar dirección del tiempo.
30    readMem(timeNow+2,10)
31    ;jmp code
32    ;nop 5
33
34  return:
35  registersymbol(timeNow)
36
37  [DISABLE]
38
39  ;timeNow:
40    ;readMem(time_bytes,10)
```

```

;db 48 B8 10 50 F6 86 F8 7F 00 00

unregistersymbol(*)
dealloc(*)

{
;ORIGINAL CODE - INJECTION POINT: 7FF887EADEF6

7FF887EADECC: 48 89 45 B8                   - mov [rbp-48],rax
7FF887EADED0: 48 89 65 90                   - mov [rbp-70],rsp
7FF887EADED4: E8 2F AE 01 FF                - call 7FF886EC8D08
7FF887EADED9: 8D B0 C8 00 00 00             - lea esi,[rax+000000C8]
7FF887EADEDF: 48 B8 F8 4F F6 86 F8 7F 00 00  - mov rax,00007FF886F64FF8
7FF887EADEE9: 33 D2                         - xor edx,edx
7FF887EADEEB: 89 10                         - mov [rax],edx
7FF887EADEED: E8 56 B1 01 FF                - call 7FF886EC9048
7FF887EADEF2: 85 C0                         - test eax,eax
7FF887EADEF4: 74 0D                         - je 7FF887EADF03
; ---------- INJECTING HERE ----------
7FF887EADEF6: 48 B8 10 50 F6 86 F8 7F 00 00  - mov rax,00007FF886F65010
; ---------- DONE INJECTING  ----------
7FF887EADF00: 83 00 0A                      - add dword ptr [rax],0A
7FF887EADF03: 48 B8 10 50 F6 86 F8 7F 00 00  - mov rax,00007FF886F65010
7FF887EADF0D: 8B 08                         - mov ecx,[rax]
7FF887EADF0F: 44 8B C1                      - mov r8d,ecx
7FF887EADF12: BA 1F 85 EB 51                - mov edx,51EB851F
7FF887EADF17: 8B C2                         - mov eax,edx
7FF887EADF19: 41 F7 E8                      - imul r8d
7FF887EADF1C: 8B C2                         - mov eax,edx
7FF887EADF1E: C1 E8 1F                      - shr eax,1F
7FF887EADF21: C1 FA 05                      - sar edx,05
```

```
{  ;  Juego  : Stardew Valley.exe
   ;  Fecha   : 2023-02-12
   ;  Este script clona los objetos del inventario, teniendo mínimo 2 objetos en el
    inventario.
}
[ENABLE]
; Patrón único en memoria.
aobscan(ITEMSTACK,AF FF 85 C0 74 0A 89 7E 44 48 83 C4 28)
alloc(newmem,$1000,ITEMSTACK)
label(code return)
alloc(auxItem,4)

auxItem:
  dd (int)999
registersymbol(auxItem)

newmem:
  ; Eliminar posibilidad de código compartido.
  cmp r9,1
  jne code
  ; Almacena nueva cantidad en edi.
  mov edi,[auxItem]
  mov [rsi+44],edi
  add rsp,28
  jmp return
```

```
code:
  ; Ejecución para las demás entidades del proceso.
  mov [rsi+44],edi
  add rsp,28
  jmp return

ITEMSTACK+06:
  jmp newmem
  nop 2
return:
registersymbol(ITEMSTACK)

[DISABLE]
ITEMSTACK+06:
  db 89 7E 44 48 83 C4 28

unregistersymbol(ITEMSTACK)
unregistersymbol(auxItem)
dealloc(*)

{
; ORIGINAL CODE - INJECTION POINT: 7FFD42ACD557

7FFD42ACD53F: 00 57 56                    - add [rdi+56],dl
7FFD42ACD542: 48 83 EC 28                 - sub rsp,28
7FFD42ACD546: 48 8B F1                    - mov rsi,rcx
7FFD42ACD549: 8B FA                       - mov edi,edx
7FFD42ACD54B: 48 8B CE                    - mov rcx,rsi
7FFD42ACD54E: E8 35 06 AF FF              - call 7FFD425BDB88
7FFD42ACD553: 85 C0                       - test eax,eax
7FFD42ACD555: 74 0A                       - je 7FFD42ACD561
7FFD42ACD557: 89 7E 44                    - mov [rsi+44],edi
7FFD42ACD55A: 48 83 C4 28                 - add rsp,28
; ---------- INJECTING HERE ----------
7FFD42ACD55E: 5E                          - pop rsi
; ---------- DONE INJECTING  ----------
7FFD42ACD55F: 5F                          - pop rdi
7FFD42ACD560: C3                          - ret
7FFD42ACD561: 3B 7E 44                    - cmp edi,[rsi+44]
7FFD42ACD564: 74 23                       - je 7FFD42ACD589
7FFD42ACD566: 48 8B CE                    - mov rcx,rsi
7FFD42ACD569: 8B D7                       - mov edx,edi
7FFD42ACD56B: E8 28 06 AF FF              - call 7FFD425BDB98
7FFD42ACD570: 48 8B CE                    - mov rcx,rsi
7FFD42ACD573: 48 B8 B8 AF 48 42 FD 7F 00 00  - mov rax,00007FFD4248AFB8
7FFD42ACD57D: 48 8B 00                    - mov rax,[rax]
}
```

A.2.2. Utilizando funciones .NET

```
{ ;  Juego  : Stardew Valley.exe
  ;  Fecha  : 2023-02-17
  ;  Este script obtiene el puntero del jugador principal para poder obtener los
    desplazamientos desde la Cheat Table.
}
```

```
5
6  [ENABLE]
7  {$lua}
8  ; En caso de no detectar la función reiniciamos los símbolos.
9  errorOnLookupFailure(false)
10 if getAddress('StardewValley.Game1::get_player+A')==0 then
11   reinitializeDotNetSymbolhandler()
12
13   if getAddress('StardewValley.Game1::get_player+A')==0 then
14     showMessage('This function is not available yet. Please try again later')
15     error() --prevents the aa entry from activating
16   end
17 end
18 {$asm}
19 ; Anclamos la función detectada desde dnSpy.
20 define(farmer,StardewValley.Game1::get_player+A)
21 ; aobscan(farmer,48 8B 00 C3 00 00 19 00 00 00 40 00 00 00 A0)
22 alloc(newmem,$1000,farmer)
23 label(code return farmer_ptr)
24 registersymbol(farmer_ptr)
25
26 newmem:
27 ; Inicializar puntero.
28 farmer_ptr:
29   dq 0
30
31 code:
32   mov rax,[rax]
33   ; Almacena el parámetro de la función en el puntero.
34   mov [farmer_ptr],rax
35   ret
36   add [rax],al
37   jmp return
38
39 farmer:
40   jmp code
41   nop
42 return:
43 registersymbol(farmer)
44
45 [DISABLE]
46
47 farmer:
48   db 48 8B 00 C3 00 00
49
50 unregistersymbol(*)
51 dealloc(*)
52
53 {
54 ; ORIGINAL CODE - INJECTION POINT: StardewValley.Game1::get_player+A
55
56 7FFCDE462AEF: 00 40 00          - add [rax+00],al
57 7FFCDE462AF2: 00 00             - add [rax],al
58 7FFCDE462AF4: 00 00             - add [rax],al
59 7FFCDE462AF6: 00 00             - add [rax],al
60 7FFCDE462AF8: 38 40 65          - cmp [rax+65],al
61 7FFCDE462AFB: DE FC             - fdivp st(4),st(0)
62 7FFCDE462AFD: 7F 00             - jg 7FFCDE462AFF
```

```
7FFCDE462AFF: 00 48 B8           - add [rax-48],cl
StardewValley.Game1::get_player+2: 70 56              - jo
    StardewValley.Menus.ColorPicker::setColor+3A
StardewValley.Game1::get_player+4: 95                 - xchg eax,ebp
; ---------- INJECTING HERE ----------
StardewValley.Game1::get_player+5: 71 2D              - jno
    StardewValley.Menus.ColorPicker::setColor+14
; ---------- DONE INJECTING  ----------
StardewValley.Game1::get_player+7: 01 00              - add [rax],eax
StardewValley.Game1::get_player+9: 00                 - db 00
StardewValley.Game1::get_player+A: 48 8B 00           - mov rax,[rax]
StardewValley.Game1::get_player+D: C3                 - ret
StardewValley.Game1::get_player+E: 00 00              - add [rax],al
7FFCDE462B10: 19 00              - sbb [rax],eax
7FFCDE462B12: 00 00              - add [rax],al
7FFCDE462B14: 40 00 00           - add [rax],al
7FFCDE462B17: 00 A0 40 65 DE FC  - add [rax-03219AC0],ah
7FFCDE462B1D: 7F 00              - jg 7FFCDE462B1F
```

```
{ ; Juego   : Stardew Valley.exe
  ; Fecha   : 2023-02-17
  ; Este script modifica la actividad de pesca haciendo uso de la función FishingRod::
    doPullFishFromWater
}

[ENABLE]
{$lua}
errorOnLookupFailure(false)
if getAddress('StardewValley.Tools.FishingRod::doPullFishFromWater+FE')==0 then
  reinitializeDotNetSymbolhandler()

  if getAddress('StardewValley.Tools.FishingRod::doPullFishFromWater+FE')==0 then
    showMessage('This function is not available yet. Please try again later')
    error() --prevents the aa entry from activating
  end
end
{$asm}
define(fish,StardewValley.Tools.FishingRod::doPullFishFromWater+FE)
; aobscan(fish,44 88 AE 83 01 00 00)
alloc(newmem,$1000,fish)
label(code return)

; Bytes
label(fish_bytes)
registersymbol(fish_bytes)

; Treasure bool      1
label(treasureCaught)
registersymbol(treasureCaught)

; Size 32            2
//label(fishSize)
//registersymbol(fishSize)

; Quality 32         3
label(fishQuality)
registersymbol(fishQuality)

```

```
; FishID 32          4
label(whichFish)
registersymbol(whichFish)

; Pond bool          5
;label(fromFishPond)
;registersymbol(fromFishPond)

; caughtDouble bool  6
label(caughtDouble)
registersymbol(caughtDouble)

newmem:
  fish_bytes:
    ; Almacenar bytes de la intrucción para prevenir futuras compilaciones indebidas.
    readMem(fish,7)
    jmp return

  treasureCaught:
    db 0 ; byte bool.
  caughtDouble:
    db 0 ; byte bool.
  fishQuality:
    dd 4 ; 32 bytes.
  whichFish:
    dd FFFFFFFF ; (-1)

code:
  mov r13l,[caughtDouble]
  mov [rbp-000000AC],r13l
  mov r13l,[treasureCaught]   ; Opción de obtener cofre o no.
  mov r15d,[fishQuality]      ; Modificar la calidad del pez (por defecto es 4).
  cmp [whichFish],FFFFFFFF    ; En caso de estar en modo deshabilitado no se modifica el
    tipo de pez.
  jz fish_bytes
  mov ebx,[whichFish]         ; Si se habilita un ID se modifica el tipo de pez.
  jmp fish_bytes

fish:
  jmp code
  nop 2

return:
  registersymbol(fish)

[DISABLE]
fish:
  ; Restaurar la zona de memoria con los bytes guardados.
  readMem(fish_bytes,7)

//db 44 88 AE 83 01 00 00

unregistersymbol(*)
dealloc(*)

{
; ORIGINAL CODE - INJECTION POINT: StardewValley.Tools.FishingRod::doPullFishFromWater+FE

```

```
 doPullFishFromWater+DB: 48 8B CF              - mov rcx,rdi
 doPullFishFromWater+DE: 48 8B 17              - mov rdx,[rdi]
 doPullFishFromWater+E1: 48 8B 52 48           - mov rdx,[rdx+48]
 doPullFishFromWater+E5: FF 52 20              - call qword ptr [rdx+20]
 doPullFishFromWater+E8: 89 85 54 FF FF FF     - mov [rbp-000000AC],eax
 doPullFishFromWater+EE: 48 8B CF              - mov rcx,rdi
 doPullFishFromWater+F1: 48 8B 17              - mov rdx,[rdi]
 doPullFishFromWater+F4: 48 8B 52 50           - mov rdx,[rdx+50]
 doPullFishFromWater+F8: FF 52 38              - call qword ptr [rdx+38]
 doPullFishFromWater+FB: 48 8B F8              - mov rdi,rax
; ---------- INJECTING HERE ----------
 doPullFishFromWater+FE: 44 88 AE 83 01 00 00  - mov [rsi+00000183],r13l  ; treasure bool
             1
; ---------- DONE INJECTING  ----------
 doPullFishFromWater+105: 44 89 B6 5C 01 00 00  - mov [rsi+0000015C],r14d ; size 32
              2
 doPullFishFromWater+10C: 44 89 BE 64 01 00 00  - mov [rsi+00000164],r15d ; calidad 32
            3
 doPullFishFromWater+113: 89 9E 60 01 00 00     - mov [rsi+00000160],ebx  ; which fish
    type      4
 doPullFishFromWater+119: 44 8B B5 58 FF FF FF  - mov r14d,[rbp-000000A8]
 doPullFishFromWater+120: 44 88 B6 87 01 00 00  - mov [rsi+00000187],r14l ; fish bond
    bool       5
 doPullFishFromWater+127: 8B 95 54 FF FF FF     - mov edx,[rbp-000000AC]
 doPullFishFromWater+12D: 88 96 88 01 00 00     - mov [rsi+00000188],dl  ; double catch
            6
 doPullFishFromWater+133: 48 8D 8E F0 00 00 00  - lea rcx,[rsi+000000F0]
 doPullFishFromWater+13A: 48 8B D7              - mov rdx,rdi
              7
 doPullFishFromWater+13D: E8 DE CD 1E 5E        - call coreclr.dll+149780
}
{
    C# Code with dnSpy
  this.treasureCaught = treasureCaught;  1
  this.fishSize = fishSize;              2
  this.fishQuality = fishQuality;        3
  this.whichFish = whichFish;            4
  this.fromFishPond = fromFishPond;      5
  this.caughtDoubleFish = caughtDouble;  6
  this.itemCategory = itemCategory;      7
}
```

```
{ ; Juego   : Stardew Valley.exe
  ; Fecha   : 2023-03-05
  ; Este script permite la recolección automática luego de plantar cualquier cultivo.
}

[ENABLE]
define(instantcrop,StardewValley.TerrainFeatures.HoeDirt::plant+121)
;aobscan(instantcrop,26 FF 49 8B 54 24 70)
alloc(newmem,$1000,instantcrop)
label(code return)
; Bytes de la instrucción.
label(hoedirtbytes)
registersymbol(hoedirtbytes)
label(end)
; Variable modificable en la Cheat Table.
label(instantreplant)
```

```
registersymbol(instantreplant)
newmem:
hoedirtbytes:
  ; Almacenamos bytes actuales;
  readMem(instantcrop,5)
  jmp return

instantreplant:
    db 0

code:
  ; mov rdx,[r12+70] Season to Grown.
  ; mov rdx,[rdx+30] Count.
  mov r9,[r12+8]    ; Crop.phaseDays (8+30+44).
  cmp r9,0
  je end
  mov r9,[r9+30]    ; Crop.phaseDays.Count.
  cmp r9,0
  je end
  mov r8d,[r9+44]   ; Crop.phaseDays.Count.Value  [32, 4bytes].
  cmp r8d,0
  jle end           ; Menor o igual 0(ZF=1).
  dec r8d

  mov r9,[r12+20]   ; Crop.currentPhase (20+ 44).
  cmp r9,0
  je end
  mov dword ptr [r9+44],r8d ; Crop.currentPhase.Value [32, 4bytes(dword)].


  mov r9,[r12+38]   ; Crop.regrowAfterHarvest (38+44).
  cmp r9,0
  je end
  mov r9d,[r9+44]   ; Crop.regrowAfterHarvest.Value [por defecto -1](4 bytes).
  cmp r9d,FFFFFFFF
  jz end

  mov r9,[r12+88]   ; Crop.fullyGrown (88+45).
  cmp r9,0
  je end
  ; cmp [instantreplant],0
  ; je end
  mov byte ptr [r9+45], 1 ; Crop.fullyGrown.Value (por defecto 0)

end:
  jmp hoedirtbytes

instantcrop:
  jmp code
  return:
registersymbol(instantcrop)

[DISABLE]

instantcrop:
  ; Restaura con valores actuales.
  readMem(hoedirtbytes,5)
  //db 49 8B 54 24 70
```

```

unregistersymbol(*)
dealloc(*)

{
; ORIGINAL CODE - INJECTION POINT: StardewValley.TerrainFeatures.HoeDirt::plant+121

 ::plant+F5: E8 FE 35 27 FF                 - call 7FFD48B4E0D8
 ::plant+FA: E9 5F 03 00 00                 - jmp  ::plant+45E
 ::plant+FF: 48 B9 28 73 CC 48 FD 7F 00 00  - mov rcx,00007FFD48CC7328
 ::plant+109: E8 82 F0 B2 5E                 - call coreclr.dll+149B70
 ::plant+10E: 4C 8B E0                       - mov r12,rax
 ::plant+111: 49 8B CC                       - mov rcx,r12
 ::plant+114: 8B D7                          - mov edx,edi
 ::plant+116: 44 8B C3                       - mov r8d,ebx
 ::plant+119: 44 8B CD                       - mov r9d,ebp
 ::plant+11C: E8 77 C6 26 FF                 - call 7FFD48B47178
; ---------- INJECTING HERE ----------
 ::plant+121: 49 8B 54 24 70                 - mov rdx,[r12+70]
; ---------- DONE INJECTING  ----------
 ::plant+126: 48 8B 52 30                    - mov rdx,[rdx+30]
 ::plant+12A: 48 B9 70 61 C7 48 FD 7F 00 00  - mov rcx,00007FFD48C76170
 ::plant+134: E8 C7 30 26 FF                 - call 7FFD48B3DBE0
 ::plant+139: 85 C0                          - test eax,eax
 ::plant+13B: 0F 84 D8 03 00 00              - je  ::plant+519
 ::plant+141: 4D 8B AE 98 00 00 00           - mov r13,[r14+00000098]
 ::plant+148: 45 39 6D 00                    - cmp [r13+00],r13d
 ::plant+14C: 49 8B CD                       - mov rcx,r13
 ::plant+14F: E8 BC 3E 27 FF                 - call 7FFD48B4E9F0
 ::plant+154: 49 8B 55 18                    - mov rdx,[r13+18]
}
```

```
{ ; Juego   : Stardew Valley.exe
  ; Fecha   : 2023-03-20
  ; Este script permite fabricar cualquier objeto especificando calidad y cantidad.
    Eliminando la lista de requisitos previos a fabricación y haciendo uso del puntero
    del ratón.
}

[ENABLE]
define(craftingrecipefull,StardewValley.Menus.CraftingPage::getContainerContents)
;aobscan(craftingrecipefull,57 56 53 48 83 EC 20 48 8B F1 48 83 BE C0)
alloc(newmem,$1000,craftingrecipefull)
label(code return)
; Bytes de la instrucción.
label(crafting_bytes)
registersymbol(crafting_bytes)
; Puntero que accede a la fabricación.
label(crafting_ptr)
registersymbol(crafting_ptr)
; Variables con punto de ruptura.
label(ptr1)
registersymbol(ptr1)
label(ptr2)
registersymbol(ptr2)
; Región cueva para realizar inyección.
label(codemode)
registersymbol(codemode)
```

```
; Salto al código original.
label(codemodepop)
registersymbol(codemodepop)
; En caso de querer fabricar un objeto.
label(EnableTypeItem)
registersymbol(EnableTypeItem)
; Almacenar el id del objeto a fabricar.
label(typeItem)
registersymbol(typeItem)

newmem:
codemode:
  ; Almacenar estado de los flags (pushf).
  ; Al mantener registros en pila y devolverlos su valor vuelve al estado original.
  push rax
  push rbx
  {$try}
  mov rax,[rcx+98]
  cmp rax,0
  je codemodepop
  mov [ptr1],rax
  mov rax,[rax+20]
  cmp rax,0
  je codemodepop
  mov [ptr2],rax
  cmp rax,0
  je codemodepop
  lea rax,[rax+38]
  cmp rax,0
  je codemodepop
  mov [rax],00000000
  ; Comparación para extraer ID.
  cmp [EnableTypeItem],FFFFFFFF
  jnz typeItem
  jmp codemodepop
  {$except}
  jmp codemodepop

typeItem: // 98 28 8 10
 mov rbx,[rcx+98]
 mov rbx,[rbx+28]
 mov rbx,[rbx+8]
 lea rbx,[rbx+10]
 mov rax,[EnableTypeItem]
 mov [rbx],rax
 jmp codemodepop

codemodepop:
  ; Devolver estado de los flags (popf).
  pop rbx
  pop rax
  jmp code

ptr2:
  dq #0
ptr1:
  dq #0
crafting_ptr:
```

```
  dq 0
EnableTypeItem:
  dd FFFFFFFF //32(-1)

crafting_bytes:
  jmp return
  //readMem(craftingrecipefull,7)


code:
  push rdi
  push rsi
  push rbx
  sub rsp,20
  ; Almacenar bytes de la instrucción.
  mov [crafting_ptr],rcx
  ; jmp crafting_bytes
  jmp return

craftingrecipefull:
  jmp newmem
  nop 2
return:
registersymbol(craftingrecipefull)

[DISABLE]

craftingrecipefull:
  db 57 56 53 48 83 EC 20

unregistersymbol(*)
dealloc(*)

{
; ORIGINAL CODE - INJECTION POINT: StardewValley.Menus.CraftingPage::getContainerContents

7FFA962D6089: 70 09                   - jo 7FFA962D6094
7FFA962D608B: C0 07 D0                - rol byte ptr [rdi],-30
7FFA962D608E: 05 E0 03 F0 01          - add eax,01F003E0
7FFA962D6093: 50                      - push rax
7FFA962D6094: 40 00 00                - add [rax],al
7FFA962D6097: 00 58 2C                - add [rax+2C],bl
7FFA962D609A: 49 96                   - xchg rax,r14
7FFA962D609C: FA                      - cli
7FFA962D609D: 7F 00                   - jg 7FFA962D609F
7FFA962D609F: 00                      - db 00
; ---------- INJECTING HERE ----------
 CraftingPage::getContainerContents: 57                      - push rdi
; ---------- DONE INJECTING  ----------
 CraftingPage::getContainerContents+1: 56                      - push rsi
 CraftingPage::getContainerContents+2: 53                      - push rbx
 CraftingPage::getContainerContents+3: 48 83 EC 20             - sub rsp,20
 CraftingPage::getContainerContents+7: 48 8B F1                - mov rsi,rcx
 CraftingPage::getContainerContents+A: 48 83 BE C0 00 00 00 00  - cmp qword ptr [rsi
    +000000C0],00
 CraftingPage::getContainerContents+12: 75 0A                   - jne  CraftingPage::
    getContainerContents+1E
 CraftingPage::getContainerContents+14: 33 C0                   - xor eax,eax
```

```
CraftingPage::getContainerContents+16: 48 83 C4 20                - add rsp,20
CraftingPage::getContainerContents+1A: 5B                         - pop rbx
CraftingPage::getContainerContents+1B: 5E                         - pop rsi
```

A.2.3. Utilizando opciones en Lua

```
{ ; Juego   : Stardew Valley.exe
  ; Fecha   : 2023-04-12
  ; Este script permite obtener una lista completa de amistades donde
   poder modificar sus atributos básicos como cantidad de regalos y
   calidad de los mismos.
}
{$lua}
-- Eliminamos registros de acciones anteriores.
if syntaxcheck then return end
for i=memrec.Count-1,0,-1 do
  memoryrecord_delete(memrec.Child[i])
end
[ENABLE]
-- Puntos asociados al NPC.
function modulePoints()
  tec = getAddressList().createMemoryRecord()
  tec.setDescription('Friendship points')
  tec.setAddress('+Friendship.points') --0008 points
  tec.OffsetCount=1
  tec.Offset[0]=0x44
  tec.VarType='vtDword'
  tec.appendToEntry(rec)
end
-- Opciones de regalo en la semana;
function moduleGiftsThisWeek()
  tec = getAddressList().createMemoryRecord()
  tec.setDescription('giftsThisWeek')
  tec.setAddress('+Friendship.giftsThisWeek')
  tec.OffsetCount=1
  tec.OffsetText[0]='NetInt.value'
  tec.VarType='vtByte'
  tec.DropDownList.Text='0:No\n1:Yes'
  tec.DropDownDescriptionOnly=true
  tec.DisplayAsDropDownListItem=true
  tec.appendToEntry(rec)
end
-- Opciones de regalo cada día.
function moduleFriendship()
  tec = getAddressList().createMemoryRecord()
  tec.setDescription('giftsToday')
  tec.setAddress('+Friendship.giftsToday')
  tec.OffsetCount=1
```

```
  tec.OffsetText[0]='NetInt.value'
  tec.VarType='vtByte'
  tec.DropDownList.Text='0:No\n1:Yes'
  tec.DropDownDescriptionOnly=true
  tec.DisplayAsDropDownListItem=true
  tec.appendToEntry(rec)
end
-- Opciones de comunicación cada día.
function moduleTalkedToday()
  tec = getAddressList().createMemoryRecord()
  tec.setDescription('talkedToToday')
  tec.setAddress('+Friendship.talkedToToday')
  tec.OffsetCount=1
  tec.OffsetText[0]='NetBool.value'
  tec.VarType='vtByte'
  tec.DropDownList.Text='0:No\n1:Yes'
  tec.DropDownDescriptionOnly=true
  tec.DisplayAsDropDownListItem=true
  tec.appendToEntry(rec)
end
-- Opciones de estados en la relación.
function moduleFriendshipStatus()
  tec = getAddressList().createMemoryRecord()
  tec.setDescription('status')
  tec.setAddress('+Friendship.status')
  tec.OffsetCount=1
  tec.Offset[0]=0x28
  tec.VarType='vtByte'
  tec.DropDownList.Text='0:Friendly\n1:Dating\n2:Engaged\n3:Married\n4:
   Divorced'
  tec.DropDownDescriptionOnly=true
  tec.DisplayAsDropDownListItem=true
  tec.appendToEntry(rec)
end
  -- Recoger puntero del jugador utilizando script.
  memrec.setAddress('farmer_ptr')  --base_ptr
  -- print(string.format("%x", memrec.getCurrentAddress()).. " farmer")
  memrec.offsetCount=4
  memrec.Offset[3]=0x520          --Friendshidata
  memrec.Offset[2]=0x30           --Dict
  memrec.Offset[1]=0x10           --entries
  memrec.Offset[0]=0x8            --Number Elements
  local count = readInteger(memrec.getCurrentAddress()) --acceder a la
   cantidad de amigos
  -- Count = 3 Size of vector

  local address = memrec.getCurrentAddress()
  -- print(string.format("%x", address).." getCurrentAddress") --Adress of
```

```
  entries
local len,pointer,name
local primeraVez = false
-- Recorremos la lista de amigos con sus atributos.
for i=0,count-2 do
  if not primeraVez then
    pointer=readPointer(address) -- Puntero
    pointer = pointer+0x10
    -- Características básicas del NPC.
    newPointer = readPointer(pointer) -- Puntero Array[0]
    -- print(string.format("%x", newPointer) .. " :pointer")
    name = readString(newPointer+0xC,256,true) -- Array[0].name
    -- print(name.." :name")

    rec = getAddressList().createMemoryRecord()
    rec.setDescription(name)
    rec.setAddress(string.format('+%X',newPointer ))--8+4 = C
    rec.OffsetCount=2
    rec.Offset[1]=0x10
    rec.OffsetText[0]='C'
    rec.VarType='vtByte'
    rec.DontSave=true
    rec.IsAddressGroupHeader=true
    -- rec.Options='[moHideChildren]'
    rec.appendToEntry(memrec)
    modulePoints()
    moduleGiftsThisWeek()
    moduleFriendship()
    moduleTalkedToday()
    moduleFriendshipStatus()
    primeraVez = true
  end
  pointer = pointer+0x18  -- Array[1+i].name
  newPointer = readPointer(pointer) -- Puntero Array[0]
  name = readString(newPointer+0xC,256,true) -- Array[0].name
  print(name.." :name")

  rec = getAddressList().createMemoryRecord()
    rec.setDescription(name)
    rec.setAddress(string.format('+%X',newPointer))--8+4 = C
    rec.OffsetCount=2
    rec.Offset[1]=0x10
    rec.OffsetText[0]='C'
    rec.VarType='vtByte'
    rec.DontSave=true
    rec.IsAddressGroupHeader=true
    --rec.Options='[moHideChildren]'
    rec.appendToEntry(memrec)
```

```
134
135       modulePoints()
136       moduleGiftsThisWeek()
137       moduleFriendship()
138       moduleTalkedToday()
139       moduleFriendshipStatus()
140   end
```

A.3. Cheats de Rising Hell

```
1  { ; Juego   : Rising Hell.exe
2    ; Versión: Build.8241641
3    ; Fecha   : 2023-04-10
4    ; Este script elimina el mensaje de detección por acceso a memoria.
5  }
6
7  [ENABLE]
8  ; Anclaje a la función anticheat.
9  define(cheatFloat,OPS.AntiCheat.Detector.FieldCheatDetector:OnCheatDetected)
10 ; aobscan(cheatFloat,00 55 48 8B EC 48 83 EC 30 48 B8 88 3A)
11 alloc(newmem,$1000,cheatFloat)
12
13 label(code)
14 label(return)
15
16 newmem:
17
18 code:
19   ; push rbp
20   ; Retorna la función.
21   ret
22   mov rbp,rsp
23   sub rsp,30
24   jmp return
25
26 cheatFloat:
27   jmp newmem
28   nop 3
29 return:
30 registersymbol(cheatFloat)
31
32 [DISABLE]
33
34 cheatFloat:
35   db 55 48 8B EC 48 83 EC 30
36
37 unregistersymbol(cheatFloat)
38 dealloc(newmem)
39
40 {
41 ; ORIGINAL CODE - INJECTION POINT: OPS.AntiCheat.Detector.FieldCheatDetector:
       OnCheatDetected
42
43 28239CE42E9: 04 02                          - add al,02
```

```
28239CE42EB: 05 04 03 01 50                 - add eax,50010304
28239CE42F0: 00 00                          - add [rax],al
28239CE42F2: 00 00                          - add [rax],al
28239CE42F4: 00 00                          - add [rax],al
28239CE42F6: 00 00                          - add [rax],al
28239CE42F8: 00 00                          - add [rax],al
28239CE42FA: 00 00                          - add [rax],al
28239CE42FC: 00 00                          - add [rax],al
28239CE42FE: 00 00                          - add [rax],al
; ---------- INJECTING HERE ----------
 FieldCheatDetector:OnCheatDetected: 55                            - push rbp
; ---------- DONE INJECTING  ----------
 FieldCheatDetector:OnCheatDetected+1: 48 8B EC                      - mov rbp,rsp
 FieldCheatDetector:OnCheatDetected+4: 48 83 EC 30                   - sub rsp,30
 FieldCheatDetector:OnCheatDetected+8: 48 B8 88 3A A5 29 82 02 00 00  - mov rax
    ,0000028229A53A88
 FieldCheatDetector:OnCheatDetected+12: 48 8B 00                      - mov rax,[rax]
 FieldCheatDetector:OnCheatDetected+15: 48 85 C0                      - test rax,rax
 FieldCheatDetector:OnCheatDetected+18: 74 1B                         - je
    FieldCheatDetector:OnCheatDetected+35
 FieldCheatDetector:OnCheatDetected+1a: 48 B8 88 3A A5 29 82 02 00 00  - mov rax
    ,0000028229A53A88
 FieldCheatDetector:OnCheatDetected+24: 48 8B 00                      - mov rax,[rax]
 FieldCheatDetector:OnCheatDetected+27: 48 8B C8                      - mov rcx,rax
 FieldCheatDetector:OnCheatDetected+2a: 48 89 45 F0                   - mov [rbp-10],rax
}
```

```
{ ; Juego   : Rising Hell.exe
  ; Versión: Build.8241641
  ; Fecha   : 2023-04-22
  ; Eliminación de la lógica para restar vida del personaje principal.
}

[ENABLE]
define(godmode,HPManager:SubtractHp)
; aobscan(godmode,55 48 8B EC 48 81 EC D0 02 00 00 48 89 7D)
alloc(newmem,$1000,godmode)

label(code)
label(return)

newmem:

code:
  ; push rbp
  ; Retorno de valor en la función.
  ret
  mov rbp,rsp
  sub rsp,000002D0
  jmp return

godmode:
  jmp newmem
  nop 6
return:
registersymbol(godmode)

[DISABLE]
```

```

godmode:
  db 55 48 8B EC 48 81 EC D0 02 00 00

unregistersymbol(godmode)
dealloc(newmem)

{
; ORIGINAL CODE - INJECTION POINT: HPManager:SubtractHp

2263F7FA2DD: 3A 26                          - cmp ah,[rsi]
2263F7FA2DF: 02 00                          - add al,[rax]
2263F7FA2E1: 00 00                          - add [rax],al
2263F7FA2E3: 00 00                          - add [rax],al
2263F7FA2E5: 00 00                          - add [rax],al
2263F7FA2E7: 00 00                          - add [rax],al
2263F7FA2E9: 00 00                          - add [rax],al
2263F7FA2EB: 00 00                          - add [rax],al
2263F7FA2ED: 00 00                          - add [rax],al
2263F7FA2EF: 00                             - db 00
; ---------- INJECTING HERE ----------
HPManager:SubtractHp: 55                                - push rbp
; ---------- DONE INJECTING  ----------
HPManager:SubtractHp+1: 48 8B EC                          - mov rbp,rsp
HPManager:SubtractHp+4: 48 81 EC D0 02 00 00              - sub rsp,000002D0
HPManager:SubtractHp+b: 48 89 7D E8                       - mov [rbp-18],rdi
HPManager:SubtractHp+f: 4C 89 75 F0                       - mov [rbp-10],r14
HPManager:SubtractHp+13: 4C 89 7D F8                       - mov [rbp-08],r15
HPManager:SubtractHp+17: 4C 8B F9                          - mov r15,rcx
HPManager:SubtractHp+1a: F3 0F 11 8D B0 FD FF FF           - movss [rbp-00000250],xmm1
HPManager:SubtractHp+22: F3 0F 11 95 A8 FD FF FF           - movss [rbp-00000258],xmm2
HPManager:SubtractHp+2a: 49 8B F9                          - mov rdi,r9
HPManager:SubtractHp+2d: 48 B8 F0 FB 9F 40 26 02 00 00     - mov rax,00000226409FFBF0
}
```

```
{ ; Juego   : Rising Hell.exe
  ; Versión : Build.8241641
  ; Fecha   : 2023-03-27
  ; Este script elimina la lógica para decrementar la cantidad de almas que pertencen al
    personaje.
}

[ENABLE]
define(almassubstract,SoulManager:DecreaseSoul)
; aobscan(almassubstract,00 55 48 8B EC 48 81 EC B0 00 00 00 48 89 7D F8)
alloc(newmem,$1000,almassubstract)

label(code)
label(return)

newmem:

code:
  ; push rbp
  ret
  mov rbp,rsp
  sub rsp,000000B0
  jmp return
```

```
23
24 almassubstract+01:
25   jmp newmem
26   nop 6
27 return:
28 registersymbol(almassubstract)
29
30 [DISABLE]
31
32 almassubstract+01:
33   db 55 48 8B EC 48 81 EC B0 00 00 00
34
35 unregistersymbol(almassubstract)
36 dealloc(newmem)
37
38 {
39 ; ORIGINAL CODE - INJECTION POINT: SoulManager:DecreaseSoul
40
41 System.Threading.Timer+TimerComparer:Compare+362: BA 55 01 00 00        - mov edx
    ,00000155
42 System.Threading.Timer+TimerComparer:Compare+367: EB E0                - jmp
    System.Threading.Timer+TimerComparer:Compare+349
43 System.Threading.Timer+TimerComparer:Compare+369: BA 6E 01 00 00        - mov edx,0000016
    E
44 System.Threading.Timer+TimerComparer:Compare+36e: EB D9                - jmp
    System.Threading.Timer+TimerComparer:Compare+349
45 2348790AA30: 01 04 02              - add [rdx+rax],eax
46 2348790AA33: 05 04 03 01 50        - add eax,50010304
47 2348790AA38: 00 00                 - add [rax],al
48 2348790AA3A: 00 00                 - add [rax],al
49 2348790AA3C: 00 00                 - add [rax],al
50 2348790AA3E: 00 00                 - add [rax],al
51 ; ---------- INJECTING HERE ----------
52 SoulManager:DecreaseSoul: 55                    - push rbp
53 ; ---------- DONE INJECTING  ----------
54 SoulManager:DecreaseSoul+1: 48 8B EC              - mov rbp,rsp
55 SoulManager:DecreaseSoul+4: 48 81 EC B0 00 00 00  - sub rsp,000000B0
56 SoulManager:DecreaseSoul+b: 48 89 7D F8           - mov [rbp-08],rdi
57 SoulManager:DecreaseSoul+f: 48 8B F9              - mov rdi,rcx
58 SoulManager:DecreaseSoul+12: 48 89 95 78 FF FF FF  - mov [rbp-00000088],rdx
59 SoulManager:DecreaseSoul+19: C7 45 80 00 00 00 00  - mov [rbp-80],00000000
60 SoulManager:DecreaseSoul+20: 48 8D 47 18           - lea rax,[rdi+18]
61 SoulManager:DecreaseSoul+24: 48 63 08              - movsxd  rcx,dword ptr [rax]
62 SoulManager:DecreaseSoul+27: 89 4D 88              - mov [rbp-78],ecx
63 SoulManager:DecreaseSoul+2a: 48 63 48 04           - movsxd  rcx,dword ptr [rax+04]
64 }
```

Bibliografía

[1] Cheat Engine, *Entorno de desarrollo enfocado en la depuración de memoria - GitHub*. [Online]. Available:https://github.com/cheat-engine/cheat-engine. [Accessed: 15-January-2023].

[2] WeMod, *Trampas y trainers para juegos de PC*. [Online]. Available:https://www.wemod.com/es/cheats. [Accessed: 18-January-2023].

[3] MrAntiFun, *Entrenadores, trucos y mods de videojuegos*. [Online]. Available:https://mrantifun.net/. [Accessed: 07-January-2023].

[4] Kaspersky, *Videojuegos: El mundo secreto de las trampas similares al malware*. [Online]. Available:https://www.kaspersky.es/blog/malware-like-cheats/19597/. [Accessed 11-January-2023].

[5] Cheat Engine, *Tipos de datos*. [Online]. Available:https://wiki.cheatengine.org/index.php?title=Tutorials:Value_types. [Accessed: 07-January-2023].

[6] Procesador Intel, *Registros Ensamblador x64*. [Online]. Available:https://www.intel.com/content/dam/develop/external/us/en/documents/introduction-to-x64-assembly-181178.pdf. [Accessed: 07-January-2023].

[7] Álvarez, J.A. and Lindig, M., *Design of Mathematical Coprocessor of Simple Precision using Spartan*, 3E Polibits, 81-90. 2008.

[8] Comandos de Cheat Engine, *Tabla de comandos*. [Online]. Available:https://wiki.cheatengine.org/index.php?title=Assembler:Commands_Tables. [Accessed: 07-February-2023].

[9] Pomeranz, Hal., *Detecting malware with memory forensics*, 2015.[Online]. Available:http://www.deer-run.com/~hal/Detect_Malware_w_Memory_Forensics.pdf. [Accessed: 07-May-2023].

[10] Poulios, Giorgos., *Postgraduate Programme Advanced Antivirus Evasion Techniques*, Master's Thesis. University of Piraeus. Greece. 2014.

[11] Team Ninja Studio, *Videojuego en línea*. [Online]. Available:https://teamninja-studio.com/nioh/. [Accessed: 11-February-2023].

[12] Tipos de plantilla, *Plantilla inyección básica*. [Online]. Available:https://wiki.cheatengine.org/index.phptitle=Cheat_Engine:Auto_Assembler#Example_using_ALLOC. [Accessed: 22-March-2023].

[13] Process-Injection, *Recursos para el TFG - GitHub Project*. [Online]. Available:https://github.com/alu0101329161/Process_Injection.git. [Accessed: 23-March-2023].

[14] Riskware, *Malwarebytes Labs*. [Online]. Available:https://www.malwarebytes.com/blog/detections/riskware. [Accessed: 23-February-2023].

[15] Tipos de juegos hackeables, *all the games you can hack with Cheat Engine*. [Online]. Available:https://www.cheatengine.org/forum/viewtopic.php?t=75249. [Accessed: 02-January-2023].

[16] Stardew Valley, *Wiki Games*. [Online]. Available:https://stardewvalleywiki.com/Stardew_Valley_Wiki. [Accessed: 12-March-2023].

[17] dnSpy, *.NET debugger and assembly editor - GitHub*. [Online] Available:https://github.com/dnSpy/dnSpy. [Accessed: 15-March-2023].

[18] Fowler Noll Vo Hash, *Algorithm Implementation*. [Online]. Available:https://thimbleby.gitlab.io/algorithm-wiki-site/wiki/fowler-noll-vo_hash_function/. [Accessed: 10-March-2023].

[19] Wireshark, *Herramienta para analizar protocolos*. [Online]. Available:https://www.wireshark.org/. [Accessed: 14-April-2023].

[20] Lidgren Network, *biblioteca de redes para .NET Framework*. [Online]. Available:https://github.com/lidgren/lidgren-network-gen3. [Accessed: 07-April-2023].

[21] Glazer, J. and Madhav, S., *Multiplayer game programming: Architecting networked games*, *Addison-Wesley Professional*, 2015.

[22] Rising Hell, *Toge Productions*. [Online] Available:https://www.togeproductions.com/project/risinghell/ [Accessed: 14-April-2023].

[23] Castillo, J.A., *Disculpas de bungie por baneos automáticos*. [Online] Available:https://www.zonammorpg.com/2022/10/09. [Accessed: 13-March-2023].

[24] League of Legends, *Página oficial del launcher*. [Online]. Available:https://www.leagueoflegends.com/es-es/. [Accessed: 13-May-2023].

[25] Riot Games, *Términos de servicios y condiciones*. [Online]. Available:https://www.riotgames.com/es-419/terms-of-service-LATAM. [Accessed: 07-March-2023].

[26] Lazarus Team, *Homepage*. [Online]. Available:https://www.lazarus-ide.org/. [Accessed: 07-January-2023].

[27] VirusTotal, *API v3 Overview*. [Online]. Available:https://developers.virustotal.com/reference/overview. [Accessed:07-January-2023].

[28] ImHex, *Hex Editor for Reverse Engineers*. [Online]. Available:https://github.com/WerWolv/ImHex. [Accessed: 07-January-2023].

[29] VPMProtect, *Software Protection*. [Online]. Available:https://vmpsoft.com/products/vmprotect/. [Accessed: 07-February-2023].

[30] Riot Games, *Reporting, Suspensions, and Bans*. [Online]. Available:https://support-leagueoflegends.riotgames.com/hc/en-us/categories/115001242847-Reporting-suspensions-bans. [Accessed: 07-February-2023].

[31] VMware, *Windows Virtual Machine*. [Online]. Available:https://www.vmware.com/products/workstation-pro.html. [Accessed: 13-February-2023].

[32] Russinovich, Mark., *DebugView - Sysinternals*. [Online]. Available:https://learn.microsoft.com/en-us/sysinternals/downloads/debugview. [Accessed: 13-May-2023].

[33] NVIDIA, *Juegos en Cloud con GeForce NOW*. [Online]. Available: https://www.nvidia.com/es-es/geforce-now/games/. [Accessed: 11-May-2023].