Aufgabe 02

Simon Kramer

May 15, 2023

1 Aufgabe

Dieses Programm ist in PowerShell geschrieben und nutzt auch die Ausführbarkeit von C-Sharp Code um einen Memory Scanner zu implementieren.

Dieser Memoryscanner soll schauen, was der Window handle, ProcessId und Process handle von dem Spiel ist und diese in der Konsole ausgeben. Danach fordert es den Nutzer auf, einen Wert einzugeben. Mit diesem Wert durchsucht es bestimmte Memory-Regionen, die zugeordnet, lesbar und beschreibbar sind. Anschließend wird der Nutzer gefordert einen (4 Byte/ 32 Integer) Wert einzugeben, mit dem der Speicher durchsucht wird. Die Memory-Regionen, welche diesen Wert enthalten, sollen in einer Hashtable niedergeschrieben und zwischengespeichert werden. Dies ist die Vorbereitung für den letzten Teil. Das Programm geht anschließend in eine Endlosschleife, welche den Nutzer erneut auffordert, einen Wert anzugeben, nachdem gescannt wird. Jedoch werden in diesem Schritt nur die überlappenden Adressen zum vorherigen Scan abgespeichert. Dies geschieht mehrfach, sodass eine Variable über mehrere Iterationen isoliert werden kann.

Weil die ganze Logik in C-Sharp implementiert ist und in PowerShell nur auf die Schnittstellen von dem C-Sharp Code zugegriffen wird, ist die PowerShell Sektionen nicht zu dokumentieren. Diese weicht auch nicht von der Vorlage ab, nur der C-Sharp Teil.

2 Code

Die erste Sektion des Codes definiert Schnittstellen zur Windows Api und die dazugehörigen DLLs. Dafür gibt man die Methoden Köpfe an und schreibt das "DllImport" Attribut davor. Zu den importierten Methoden gehören:

- **FindWindow:** ermöglicht ein Fenster nach dem Namen zu suchen und einen Window-Handle zu erhalten
- **GetWindowThreadProcessId:** ermöglicht eine ProcessId zu bekommen basierend auf einen Window-Handle

- **OpenProcess:** ermöglicht einen Process-Handle zu bekommen basierend auf einer ProcessId
- **ReadProcessMemory:** ermöglicht Speicher, der von einem bestimmten Prozess benutzt wird, zu lesen.
- **VirtualQueryEx:** ermöglicht Informationen über Speicherregionen abzufragen

Zudem wird in diesem Teil die Struktur von Mbi (MemoryBasicInformation) definiert. Für ausführliche Informationen auf https://pinvoke.net/index.aspx ist alles nochmal sehr ausführlich dokumentiert.

2.1 Initialisierung

Am Anfang des Programms, erstellen wir eine Instanz und übergeben den Namen des Fensters, das zum Prozess gehört, den wir scannen wollen.

Im Konstruktor werden danach die Anfragen zur Windows-Api getätigt und die Ergebnisse werden in die Properties "WindowHandle" "ProcessId" und "ProcessHandle" geschrieben. Des Weiteren werden auch nur die Memory-Regionen gespeichert, die für uns von Bedeutung sind. Das bedeutet, dass die Property "Proect" besagt, dass die Region les- und schreibbar ist und die Property "State" "MEM-COMMIT" ist. Dies wird durch eine (Hexadezimal) Zahlen angegeben.

2.2 ReadBuffer

Die "ReadBuffer"-Methode erwartet ein Byte Array und gibt ein Integer Array zurück. Die Funktion funktioniert indem sie über den Buffer geht und alle möglichen Kombinationen von 4 Bytes zu einer 32Integer konvertiert, danach werden diese in einem Array von Integer Werten abgespeichert. Es ist nicht möglich in vierer Schritten über die Speicherregionen zu gehen, weil Datentypen unterschiedlich groß seien können. Die unterschiedlichen Größen führen dazu, dass man nicht weiß wo eine Integer anfängt und wo sie aufhört. Deshalb muss man alle möglichen Kombinationen von Bytes betrachten.

2.3 ScanAddress

Die "ScanAddress"-Methode erwartet eine 64Integer und gibt eine 32Integer zurück. Innerhalb der Methode wird ein Buffer erstellt, in den 4 Bytes von der gegebenen Adresse geladen werden. Diese werden dann mit der "ReadBuffer"-Methode zu einer Integer konvertiert und zurück gegeben.

2.4 ScanRegion

Die "ScanMemory"-Methode erwartet einen Mbi-Struct und ein Integer-Wert. Das Mbi-Struct definiert die Memory Region, die gescannt werden soll. Der Integer Wert definiert einen Wert, nach dem gesucht wird.

Die Methode funktioniert indem sie in jeder Iteration eine Speicher Sektion scannt, die Werte werden in einen Buffer geschrieben, der mit ReadBuffer in ein Integer Array konvertiert wird. Danach wird das Integer Array überprüft, ob dort Werte enthalten sind, die gleich dem Ziel Wert sind. Wenn ein Wert gleich dem Zielwert ist, dann wird die Speicheradresse als Schlüssel der Hashtable genutzt und die Value als gespeicherten Wert.

Danach wird die Laufvariable um den Wert erhöht, um wie viele Bytes gelesen wurden.

Das wird solange durchgeführt, bis entweder die Länge der gelesenen Bytes 0 ist oder die insgesamt gelesenen Bytes größer gleich der Speicherregiongröße ist.

2.5 ScanProcess

Die "Scan"-Methode erwartet einen Integer-Wert und gibt eine Hashtable zurück. Innerhalb der Methode wird für alle Mbis die "ScanMemory"-Methode aufgerufen. Die unterschiedlichen Hashtables werden danach zu einer großen zusammengefasst und zurück gegeben.

2.6 ValidateScan

Die "ValidateScan"-Methode nimmt eine Hashtable, einen Zielwert (Target) entgegen und gibt eine Hashtable zurück. Die Methode fragt mit "ScanAddress" alle Memory Adressen ab, die in der Hashmap gegeben sind. Sie vergleicht die Werte, die bei den Adressen hinterlegt sind, mit dem Zielwert. Wenn der Zielwert übereinstimmt, wird dies in die Hashtable "result" geschrieben und zurück gegeben.