Analiza Vulnerabilităților Software Într-o Aplicație Unity

Darius-Alex Roșca

March 31, 2025

1 Obiectivul Proiectului

Scopul principal al proiectului este de a descoperi anumite vulnerabilități într-o aplicație Unity pentru ca atacatorul să le poată folosi în favoarea lui. De asemenea, odată descoperite, se vor rezolva acele vulnerabilități.

2 Obiectivul Jocului

Jocul dezvoltat în Unity, versiunea 2022.3.41f1, prezintă o serie de caracteristici implementate, respectiv: viața, viteza de atac, viteza de mișcare, numărul de monștri creați la fiecare nivel și numărul de monștri omorâți. Scopul jocului este ca jucătorul sa reușească să omoare cât mai mulți monștri.

3 Metode si Materiale

3.1 Cheat Engine

Cheat Engine este o aplicație care facilitează accesul la memorie pentru analizarea și modificarea datelor dintr-un joc sau o aplicație în timp real. Mai exact, aplicația scanează memoria RAM, permițând astfel modificarea variabilelor [Erind].

1. **Atac** - O primă vulnerabilitate găsită este legată de viața jucătorului. O dată lansat jocul, se poate observa viața curentă a jucătorului, în acest caz: 100. Odată știut acest lucru, se poate face prima căutare în memorie folosind Cheat Engine, pentru a vedea toate adresele care au în interior salvate variabila 100. În continuare va fi selectat procesul jocului:

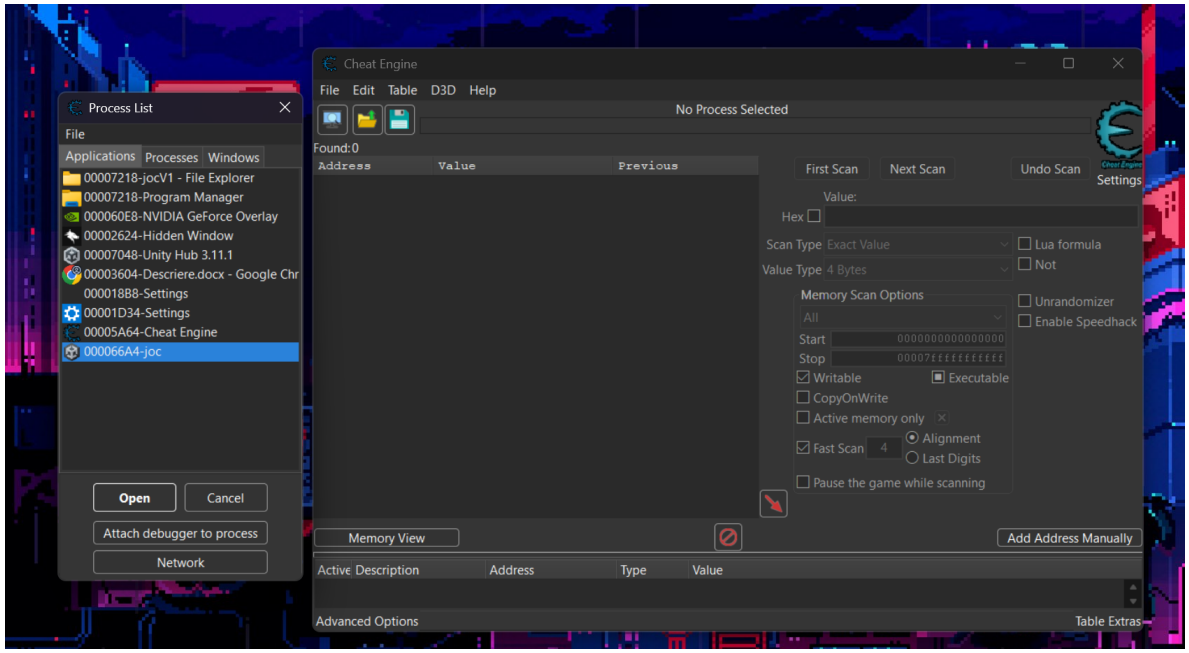

Figure 1: Selectarea procesului

Odată selectat se poate realiza prima scanare. In figura 2 se va ilustra acest lucru.

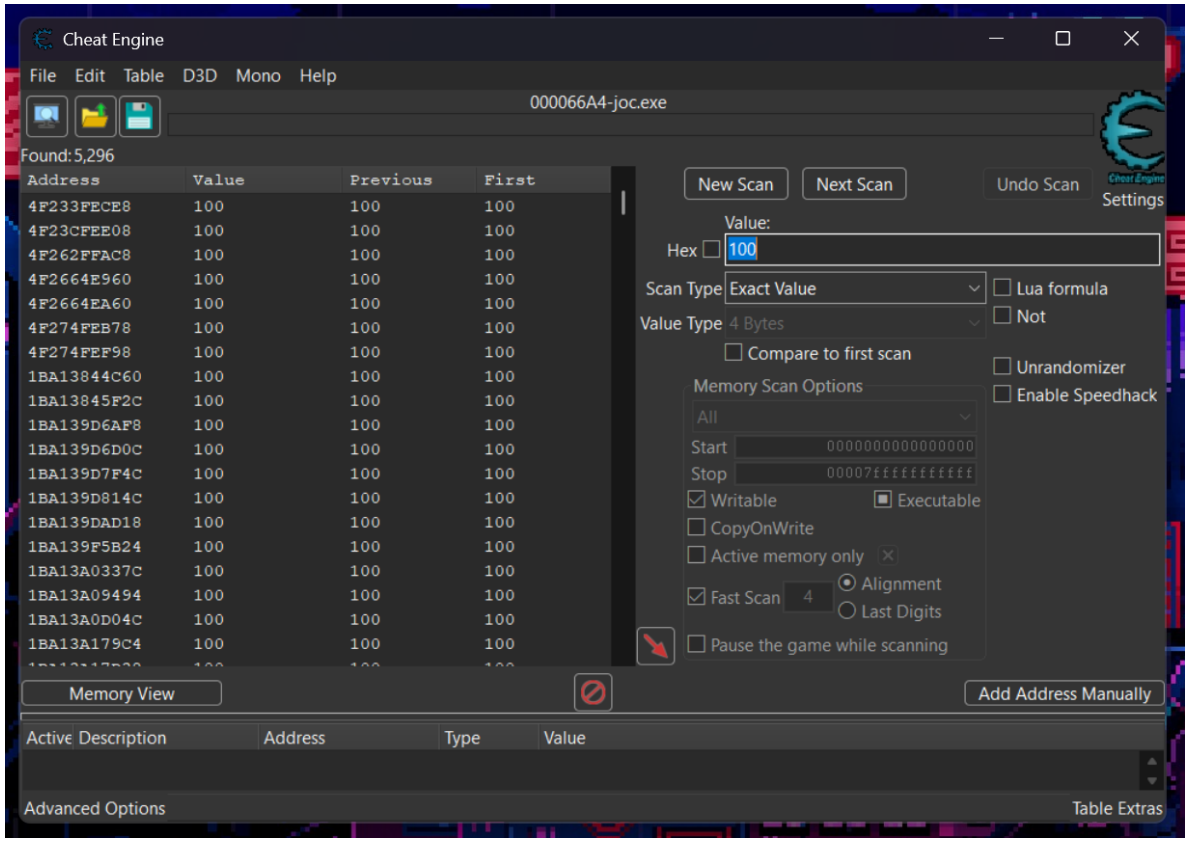

Figure 2: Prima scanare

In prima scanare, au fost căutate toate adresele care conțin valoarea 100 pe 4 Bytes. Rezultatul obținut a fost de 5296 de adrese. În acest caz, nu este posibilă determinarea adresei corecte, astfel, pentru a reduce numărul de adrese, se va continua cu altă scanare.

A doua scanare este realizată prin scăderea vieții jucătorului de la 100 la 70.

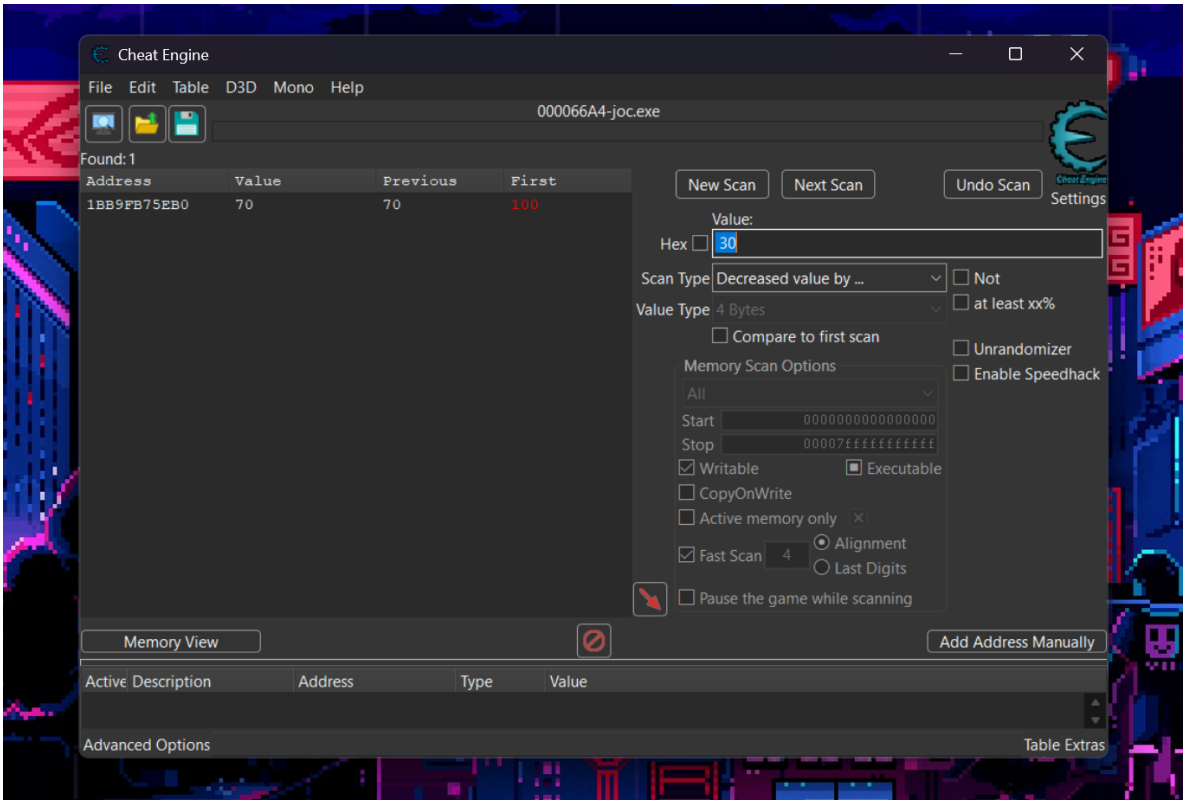

Figure 3: A doua scanare

Odată scăzută viața, din cele 5296 de adrese, au fost căutate doar adresele a căror valoare a scăzut cu 30 de unități, astfel, s-a obținut o singură adresă. Pentru a avea o certitudine legată de faptul că adresa găsită este cea corectă, se poate scădea din nou viața jucătorului, observând astfel dacă și valoarea adresei găsite va scădea exact cu aceeași valoare. Prin urmare, viața jucătorului a fost scăzută la 50 de unități.

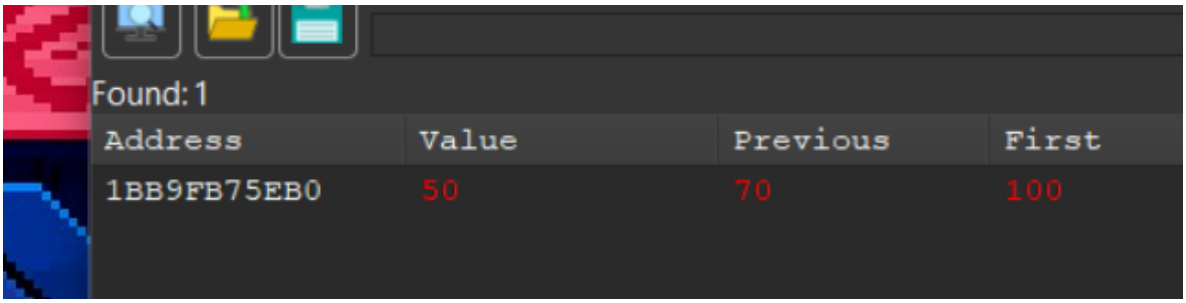

Figure 4: Rezultatul scăderii

Odată ce valoarea din adresă a scăzut la 50 de unități, se poate modifica pentru a observa dacă se obțin rezultatele dorite în joc.

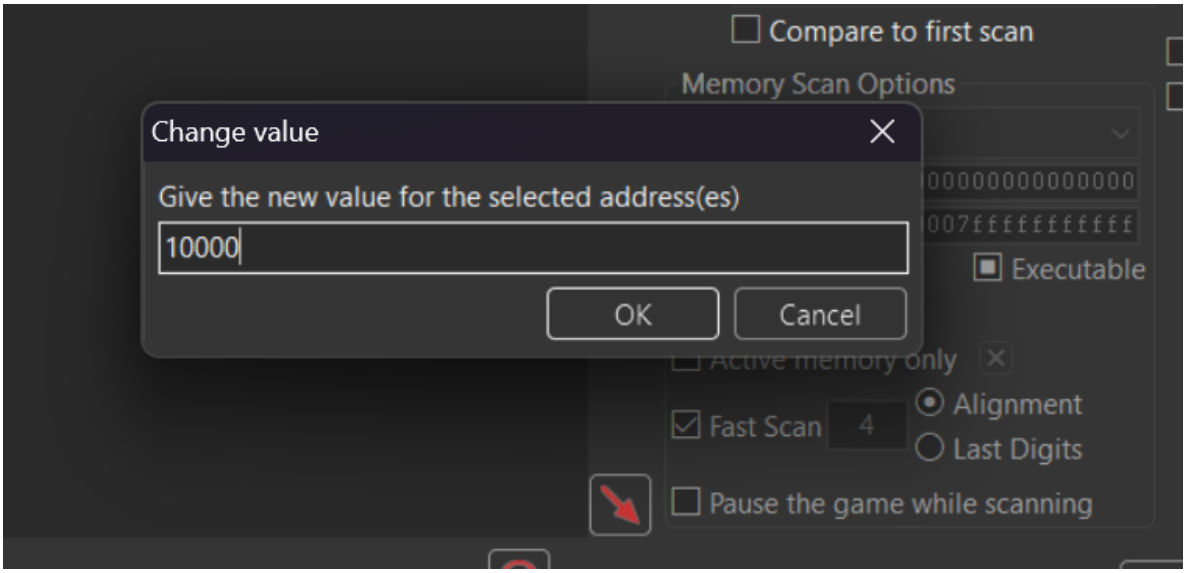

Figure 5: Modificarea valorii adresei

În următoarea etapă, s-a modificat valoarea din adresă, respectiv de la 50 la 10000. În figura 5 se observă modificarea valorii.

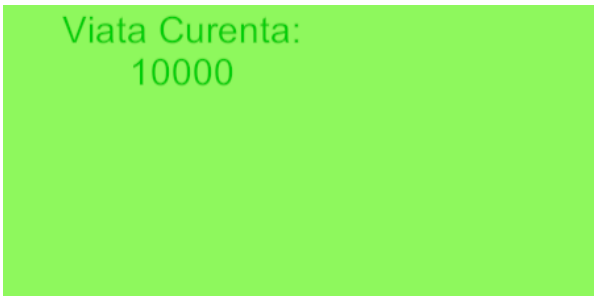

Figure 6: Afișarea vieții curente în joc

În figura 6, se ilustrează faptul că modificările aduse se reflectă și în joc. In cazul celorlalte atribute din joc, descoperirea lor a decurs in mod similar.

2. **Soluție** - O soluție simplă, dar ineficientă, este de a seta un prag maxim pentru atribute, iar verificarea acestora să fie efectuată periodic. Însă, atacatorul poate determina acest lucru rapid, aducând modificări minore asupra valorilor pentru a nu depăși pragul.

   O soluție eficientă este de a cripta atributele și de a salva doar varinta lor criptată. Algoritmul de criptare ales este AES doarece performanța jocului nu va fi modificată datorită complexității acestuia.

   Pasul următor conține implementarea clasei AESHelper. Această clasă este dezvoltată atât cu ajutorul bibliotecii *System.Security.Cryptography*, cât și cu ajutorul exemplului [Micnd].

   Odată implementată această tehnică de criptare și de decriptare, s-au efectuat în mod repetat diferite teste.

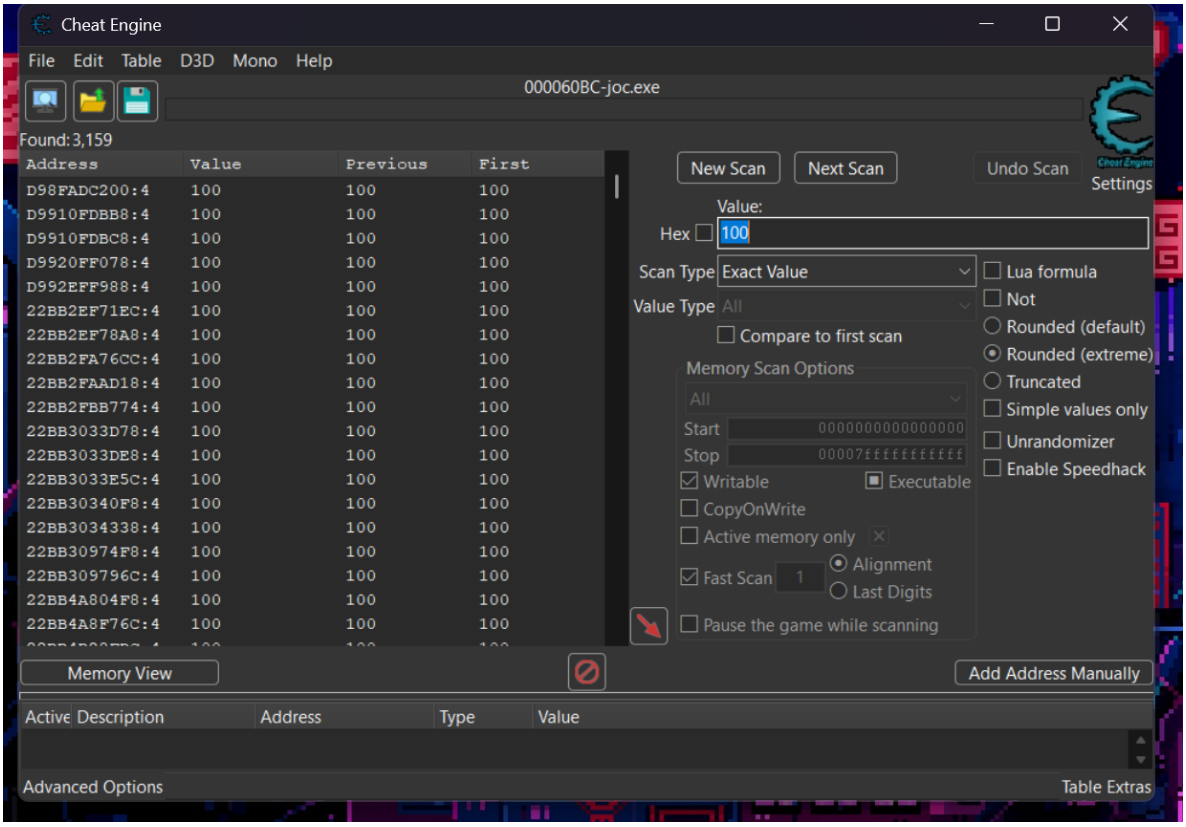

Figure 7: Scanare

În continuare, se vor căuta toate adresele care conțin valoarea 100, urmând mai apoi ca viața jucătorului să fie scăzută la 70.

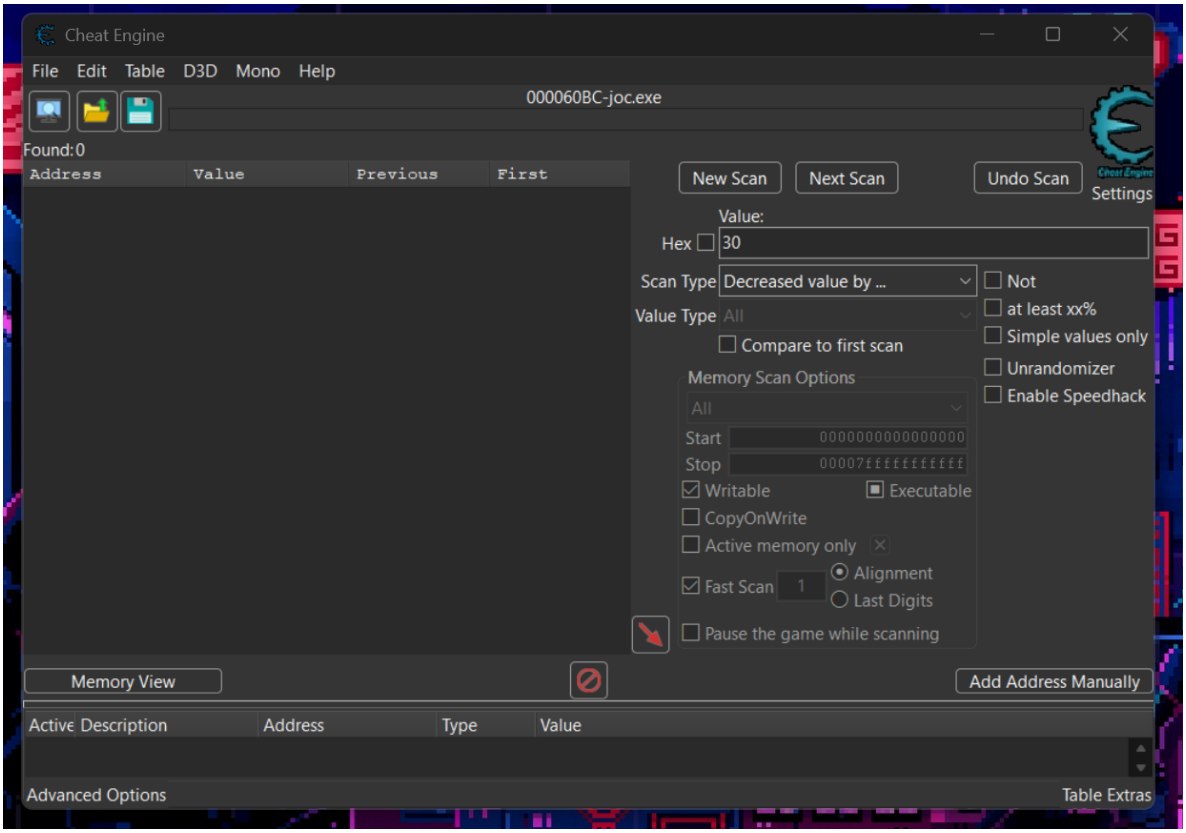

Figure 8: Rezultatul scăderii

Odată scazută viața, din cele 3159 de adrese găsite, s-au căutat adresele care conțin valoarea scăzută cu 30 de unități. Rezultatul acestei scanări este unul negativ, deoarece nu a fost găsită nicio adresă care să respecte condiția de mai sus.

Presupunând faptul că atacatorul intuiește că atributele au fost ciptate, acesta poate căuta adresele care reține varabilele de tip String sau Array Bytes pentru a determina anumite atribute precum: cheia, IV ul sau criptul. Însă, adresele care țin aceste tipuri de variabile sunt numeroase si relativ greu de urmărit doarece acestea nu prezintă o creștere sau scădere lineară. Din cauza acestui lucru, nu se pot realiza anumite conexinui logice între joc și valorile reținute în adrese.

Având în vedere acest fapt, atacatorul nu poate modifica adresele corespunzatoare sau să folosească tool-uri, precum: Debugging sau Assebly, tool-uri care sunt puse la dispoziție de către aplicație pentru a manipula interacțiunea cu ele, până când nu găsește adresele.

3.2 dnSpy

dnSpy este o aplicație Open-Source folosită pentru decompilare, editare și depanarea aplicațiilor care folosesc C# facilitând ingineria inversă a aplicațiilor dezvoltate cu .NET si Unity [dnS23].

1. **Atac** - O altă vulnerabilitate este legată de modul în care este compilată aplicația.

   În mod implicit, Unity folosește Mono pentru compilarea și rularea codului C#. Cum funcționează Mono:

   In timpul dezvoltării, codul C# este convertit în CIL (Common Intermediate Language) și stocat în fișiere .dll, cum ar fi: Assembly-CSharp.dll.

   In timpul execuției, Mono interpretează și compilează codul CIL în cod mașină nativ printr-un proces numit JIT (Just-In-Time Compilation).

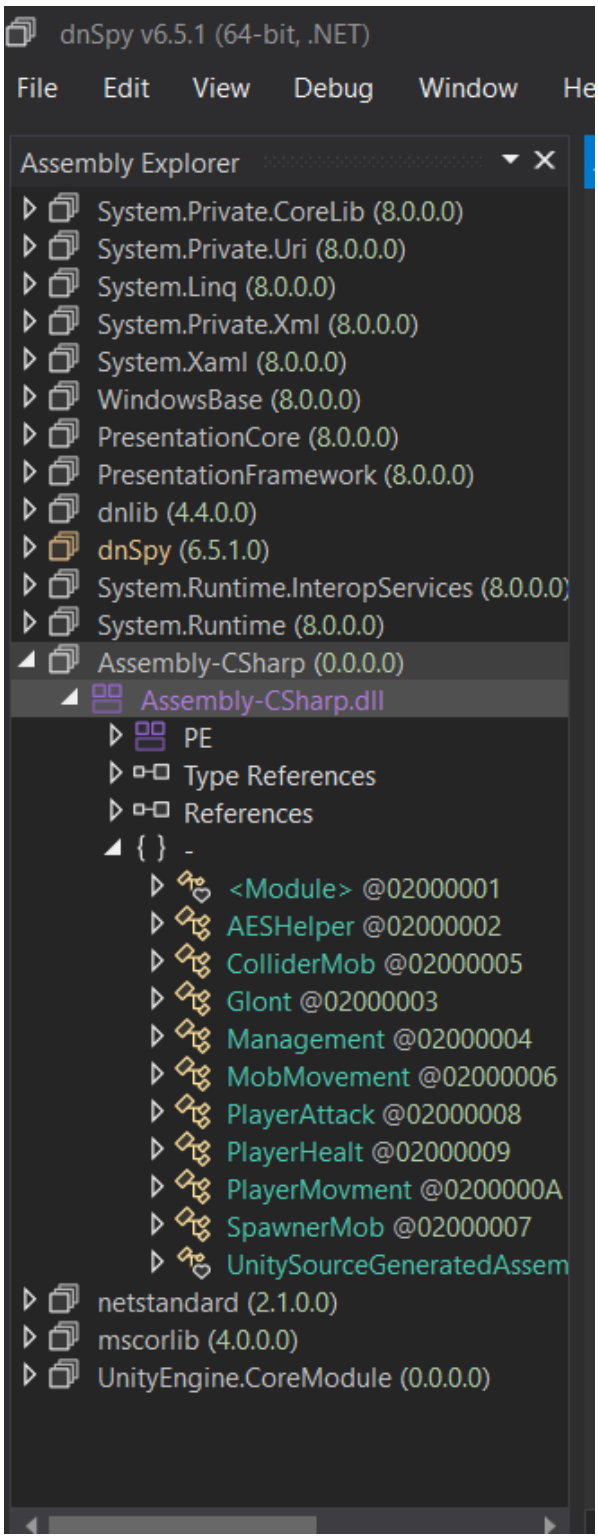

Figure 9: Accesul la clasele proiectului

Odată deschis, fisierul Assembly-CSharp.dll, din cadrul proiectului, se permite accesul la clasele acestuia.

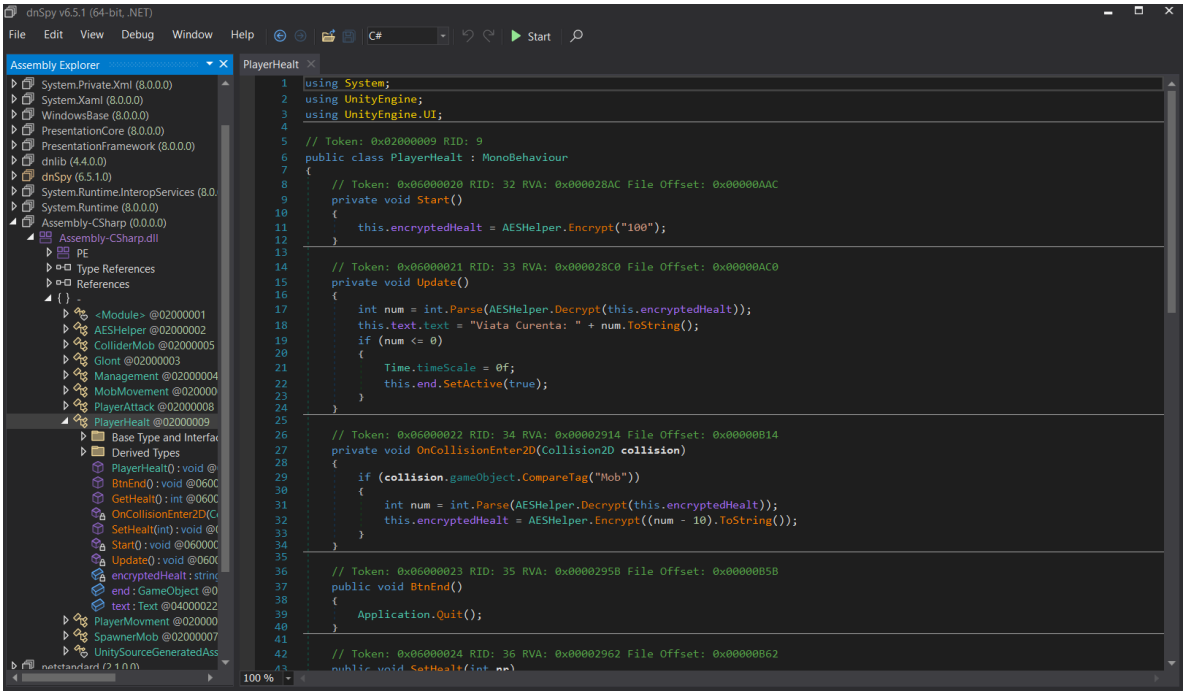

Figure 10: Decompilarea clasei

La deschiderea orecărei clase, dnSpy va decompila automat clasa, având astfel acces la întregul cod scris. In acest exemplu, am deschis clasa PlayerHealt în care se poate observa o metodă numită OnCollisionEnter2D. Acestă metodă se ocupă cu gestionarea interacțiunii jucătorului cu alte obiecte din joc. In acest caz, în momentul în care jucătorul face contact cu un inamic, pierde 10 uități din atributul viață.

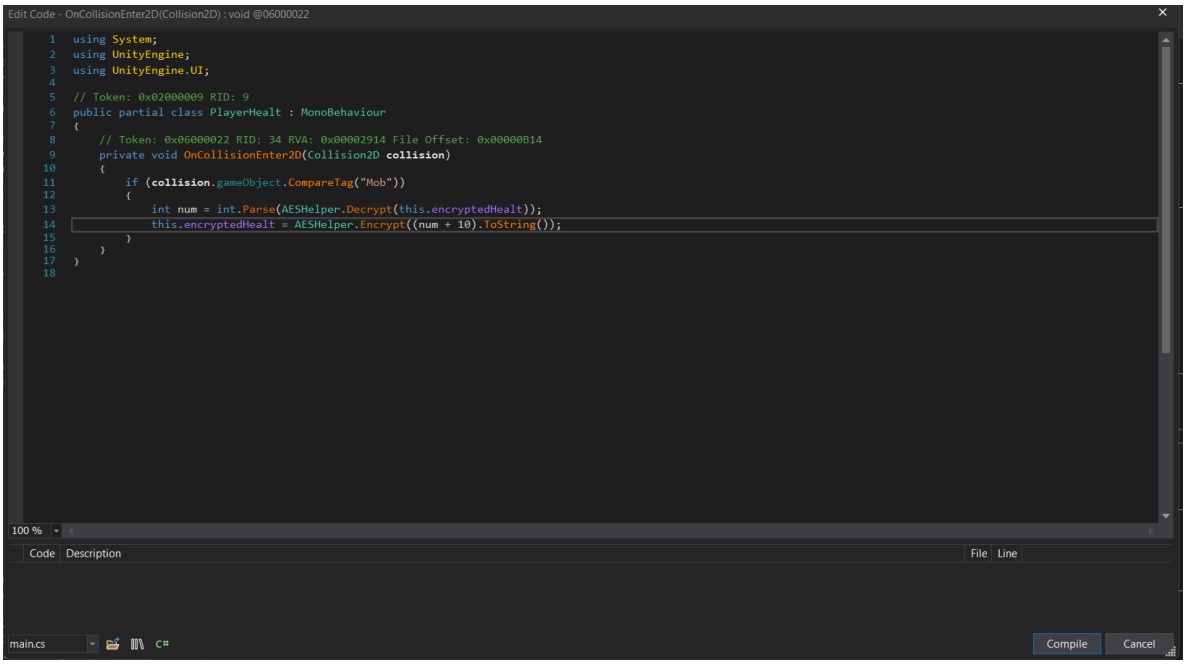

Figure 11: Modificarea metodei

Se va modifica metoda pentru ca jucătorul să primească 10 uități de viață cand va face contact cu inamicii.

Figure 12: Vizualizarea modificării în joc

Odată aplicată modificarea, jucătorul va fi invincibil.

2. **Soluție** - O soluție temporală împotriva ingineriei inverse este de a schimba modul de compilare a priectului, respectiv, în loc să se folosească Mono, se va folosi IL2CPP.

   Ce face IL2CPP:

   Spre deosebire de Mono, în timpul rulării, IL2CPP adaugă un pas în plus, convertind astfel CIL-ul în C++. În timpul execuției, codul generat este compilat în cod nativ pentru platformele țintă.

4 Concluzie

În acest studiu au fost analizate o serie de vulnerabilități Software într-o aplicație Unity. S-au folosit atât tehnici de scanare a memoriei, cât și tehnici de inginerie inversă. Pe viitor, acest proiect se poate extinde prin exploatarea a mai multor vulnerabilități cât și rezolvarea acestora.

References

[dnS23] dnSpy. dnspy - .net decompiler, editor, and debugger, 2023. Accessed: 31-Mar-2025.

[Erind] Heijnen Eric. About cheat engine, n.d. Accessed: 31-Mar-2025.

[Micnd] Microsoft. System.security.cryptography.aes class, n.d.