Laboratorul nr. 7
Analiza de executabil utilizând Ghidra

Pentru a testa un program de acest tip se poate utiliza pentru un începător un executabil creat în C și apoi în CPP pentru un program simplu. După deprinderea abilităților de bază procesul se va relua pe programe de același tip ceva mai complicate și în final se poate trece pe analiza unui executabil despre care nu avem deloc conștiințe. Evident că trebuie reîmprospătate înainte conștiințele de ASM cu orientare pe compilatorul specific LINUX (vezi și cursul)

Pentru a instala Ghidra este nevoie de JDK 11. Atenție la situația în care avem nevoie în funcție de tipurile de aplicații instalate de mai multe JDK-uri simultan.

Apoi aceasta se aduce de la https://ghidra-sre.org/ghidra_9.0.4_PUBLIC_20190516.zip

În cazul unor dificultăți în cadrul procesului de instalare se va consulta https://www.ghidra-sre.org/InstallationGuide.html#Install.

În directorul utilizatorului se creează (dacă nu exista directorul opt în care apoi se despachetează arhiva. Se va executa cu ./ghidraRun de la consolă sau se poate crea un shortcut la alegere.

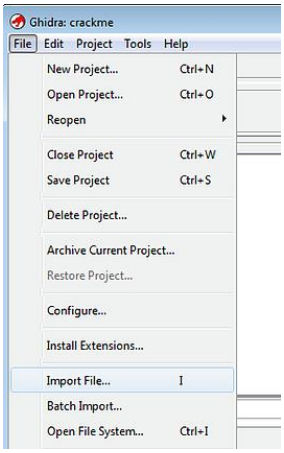
Ce este Ghidra?

Deși la curs am prezentat o serie de alternative de instrumente pentru analiza executabilelor nu am detaliat mult Ghidra deoarece a fost aleasă ca aplicați pentru laborator. Aceasta este un instrument gratuit, relativ nou în piața utilitarelor de profil, care pornește de la un proiect mai vechi al NSA numit “Vault 7”.

Aplicația poate fi executată pe toate sistemele de operare dominante și conține toate abilitățile necesare analizei și modificării unui executabil oarecare. Aceste aspecte au fost analizate la curs deci se presupun a fi cunoscute. O caracteristică mai rar întâlnită la programele de acest tip constă în faptul că permite o abordare cooperativă a unei echipe în desfășurarea analizei pentru același program.

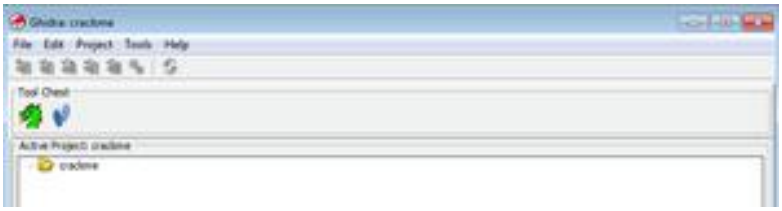
În laborator se vor testa numai abilitățile de bază ale aplicației.

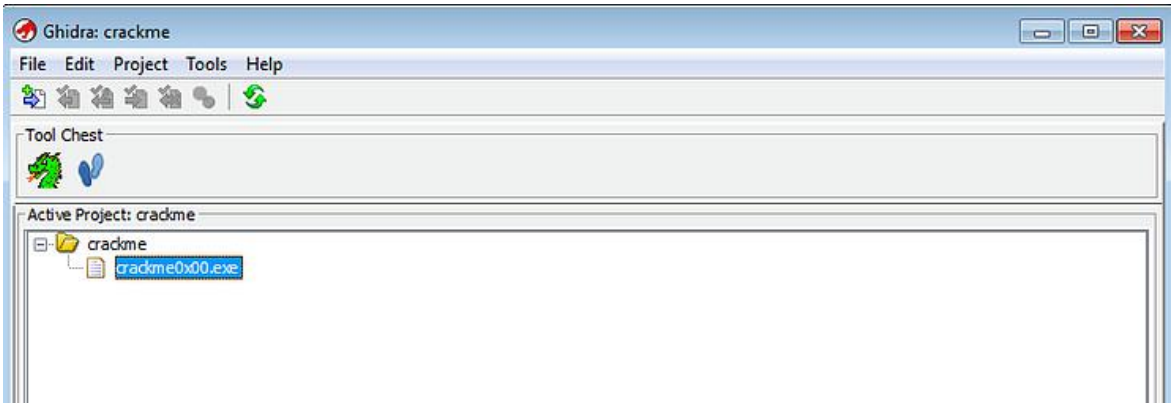
Crearea unui proiect



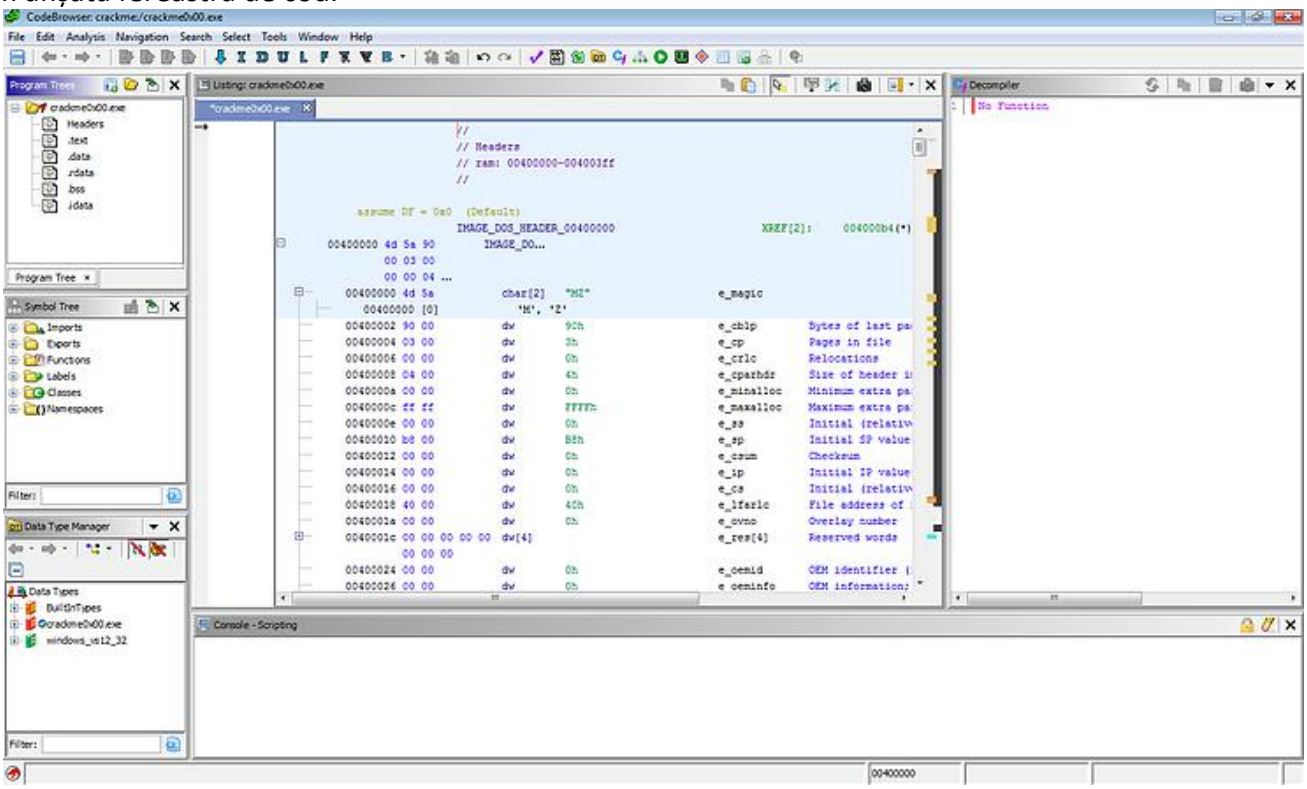
După cum am spus la început se recomandă analiza unor executabile mai simple. Totuși pentru a pune în evidență abilitățile primare ale aplicației s-a folosit un executabil ceva mai complicat numit crackme0x00.exe care este disponibil la (https://github.com/Maijin/Workshop2015/tree/master/IOLI-crackme/bin-win32).

Se va crea un nou proiect și se poate trage direct executabilul din double commander în fereastra acestuia sau se poate apela la maniera clasică: File -> Import File oferită de opțiunile proiectului.





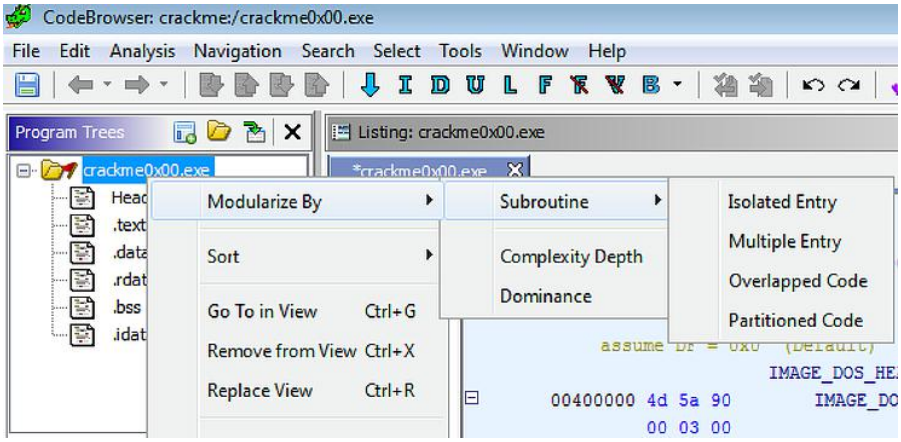
Apoi cu un dublu-click pe crackme0x00.exe se va deschide navigatorul de cod și va apare întrebarea: *"Do you want to analyze the binary file?"* Odată acceptată se vor prezenta mai multe opțiuni cu privire la analiza excutabilului. În această fază vom alege *"Analyze"*. După terminarea operațiunii va fi afișată fereastra de cod.



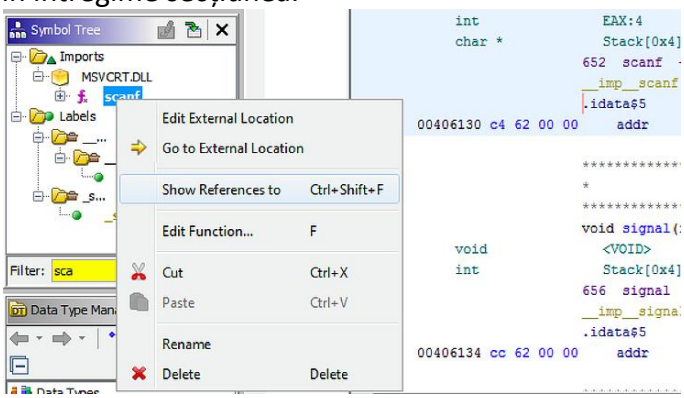
Fereastra Principală

Primul lucru observat (și destul de rar) este faptul că oferă help adaptat la context prin apăsarea lui F1 când mouse-ul este deasupra unei entități din meniu.

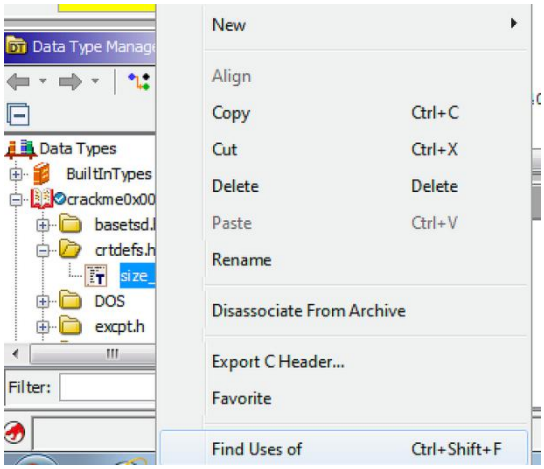
Cu ajutorul ferestrei *"Program Trees"* și a unui click dreapta peste directorul crackme0x00.exe se pota selecta diverse opțiuni de analiză inversă a codului. De exmplu se poate merge pe următoarea suită de comenzi: "Modularize By" -> "Subroutine" sau "Complexity depth" sau "Dominance". Programul permite crearea de noi directoare și tragerea directă cu mouse-ul conform necesităților utilizatorului.



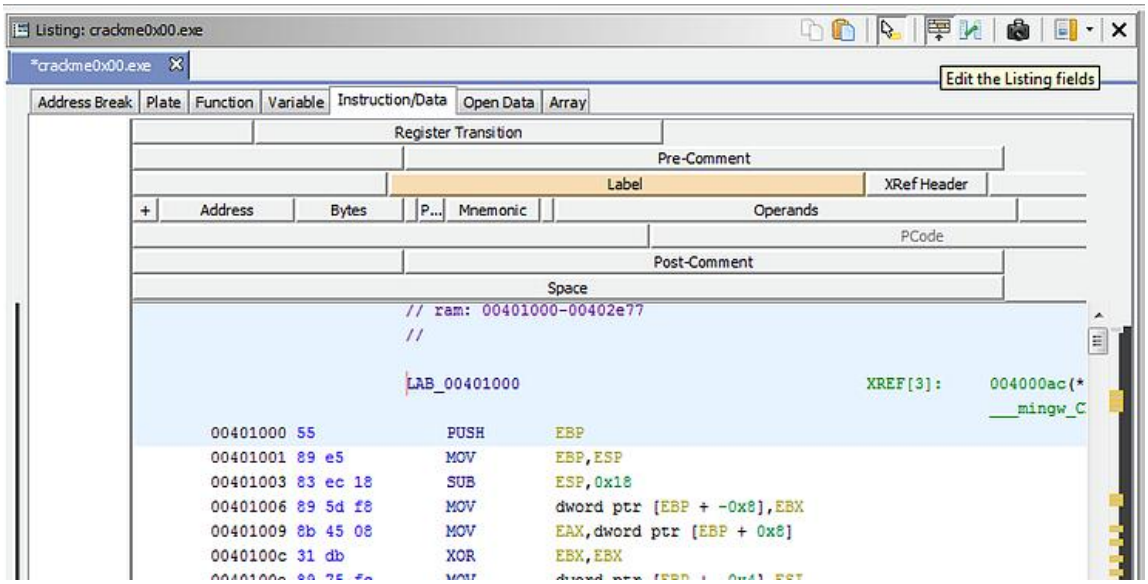
Următoarea fereastră sub "Program Trees" este "Symbol Tree" care ne permite vizualizare, importul, exportul, funcții, etichete, clase și namespace-uri pentru un fișier executabil. Încercați să activați secțiunea "Import" pentru a vedea diverse DLL-uri și funcțiile lor. Dacă doriți să se vizualizeze locul unde sunt plasate diverse funcții în executabil se poate dat click dreapta și apoi dublu click pe rezultat pentru a vedea în întregime secțiunea.



Opțiunea "Data Type Manager" ne permite să vizualizăm tipuri specifice inclusiv cele create de utilizator care sunt comune unui utilizator inclusiv pe cel incluse în Ghidra (de ex cele din fereastra "windows_vs12_32"). Încercați să despachetați icoanele carte prin click dreapta asupra opțiunii Data Type. Apoi click pe "Find Uses of" pentru a putea vedea unde este utilizat respectivul tip de date în executabil.

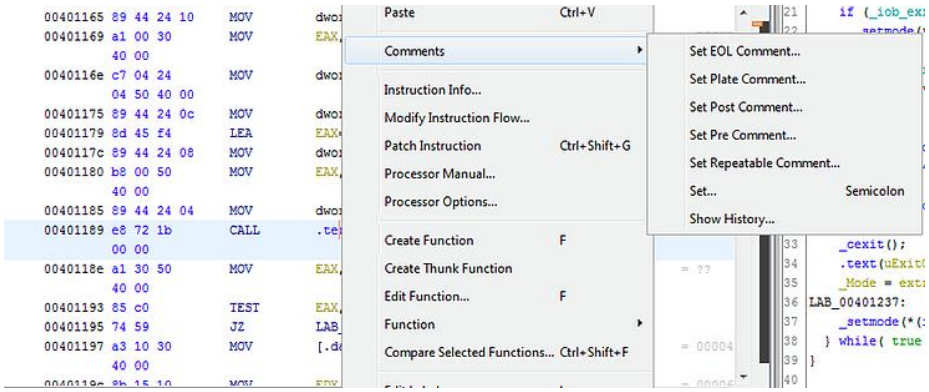


Cu ajutorul ferestrei “Listing” se poate inspecta rezultatul dezasamblării codului. Ghidra ne oferă mute opțiuni în configurarea listbox-ului. Pentru aceasta se va utiliza iconița “Edit the listing fields” (aflată în partea dreaptă sus) apoi se va selecta tab-ul “Instruction/Data”. Orice element al interfeței poate fi schimbat, mutat, dezactivat sau eliminat. De asemenea se pot adăuga noi elemente prin click dreapta și utilizarea meniului context. Încercați să modificați câmpul “Address” în sensul reducerii dimensiunii și eliminați câmpul de octeți - “Bytes”.

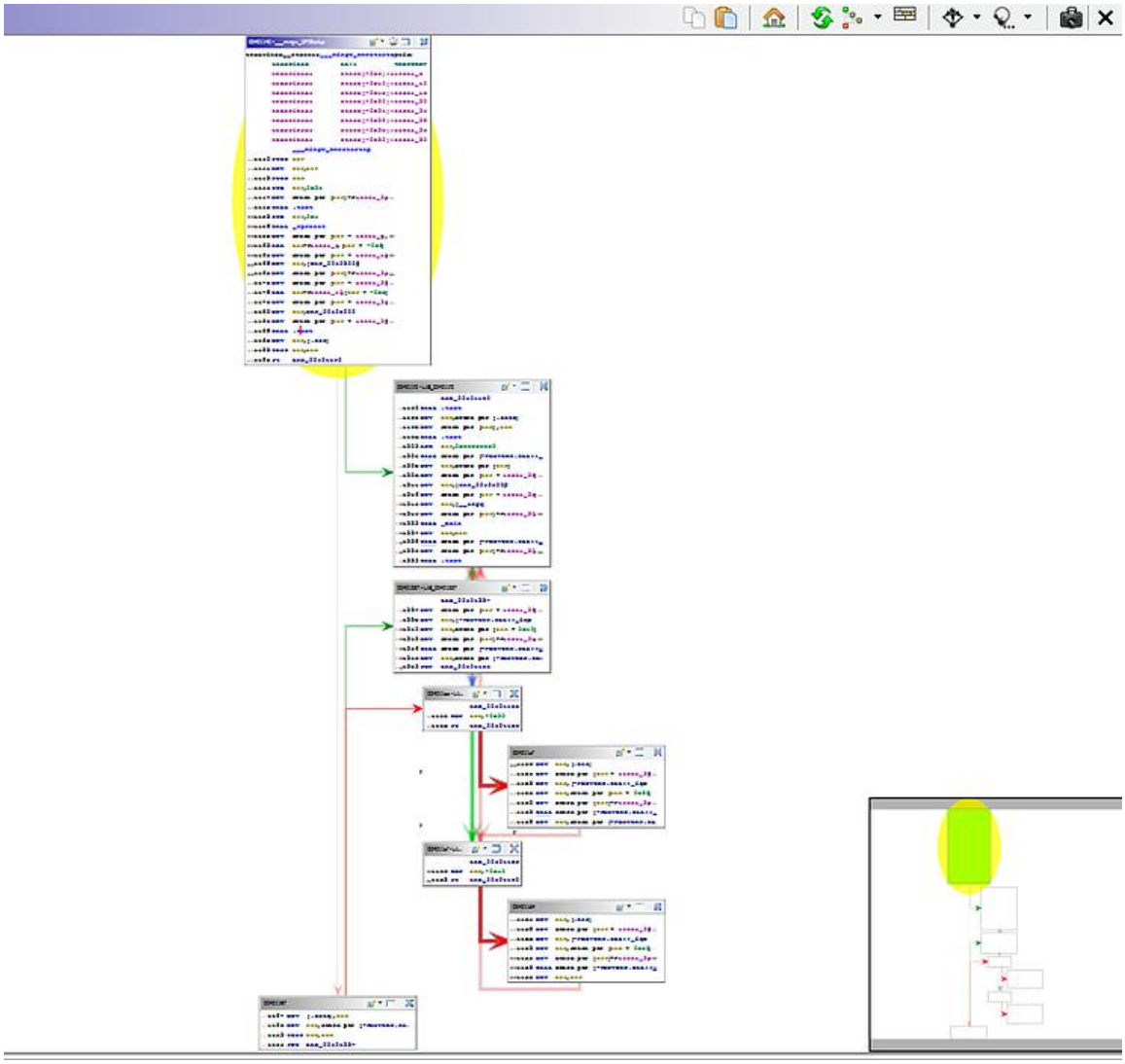


Veți vedea un nou meniu context în sursa ASM pentru analiză inversă prin click dreapta oriunde în fereastra care conține codul în limbaj de asamblare. Aici se pot corecta instrucțiuni, stabili bookmark-uri și editare de scurtături. Încercați să adăugați un comentariu cu ajutorul lui click dreapta pe una dintre instrucțiuni.

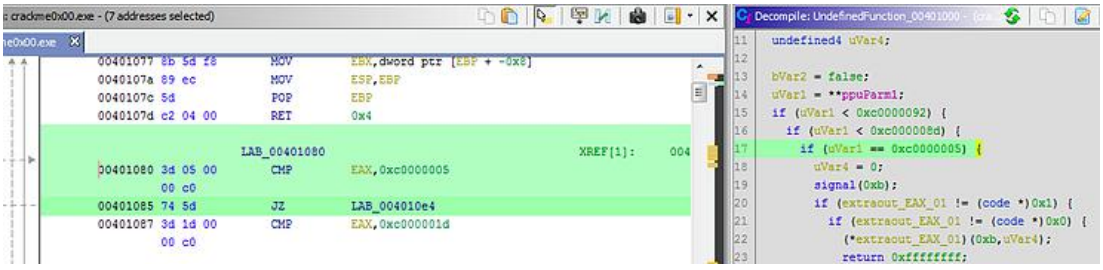
Dacă se face un dublu click pe una din funcțiile indicate de “CALL” vom fi trimiși la codul ei. Navigați utilizând pe săgețile din stânga sus cu ajutorul icoanei “save” sau a combinațiilor “Alt-Left Arrow Key” și “Alt-Right Arrow Key” .



Pentru crearea CFC-urilor (vezi curs) Ghidra oferă “Function Graph”, care poate fi accesată prin suita de comenzi “Window” -> “Function Graph”. La ea se mai poate ajunge și prin intermediul butonului “Edit the listing fields”. Acest grafic nu include comentarii dar permite adăugarea acestora dupa necesități cu ajutorul lui Field Editor.



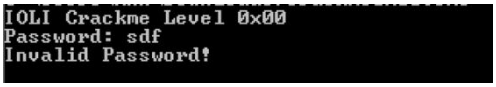
În sfârșit vom vedea fereastra “Decompile” care afișează în partea dreaptă un cod de nivel înalt generat de Ghidra și care este asociat codului ASM rezultat din dezasamblare. În momentul în care se va marca o instrucțiune din acest cod automat va fi marcat și set de instrucțiuni ASM corespunzător ei.



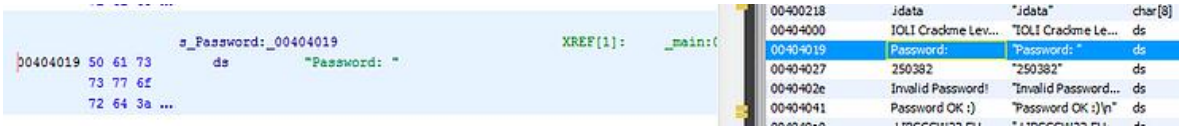
Numele variabilelor poate fi schimbat sau comentarii pot fi adăugate (cu ajutorul unei opțiuni date de click - dreapta). Acestea vor fi afișate și în fereastra de “Listing”. Dacă se dorește stabilirea unor parametri care trebuie afișa și se va face click dreapta în fereastra și se va selecta “Properties”. Verificați dacă puteți denumi de exemplu o variabilă locală utilizând un nume mai apropiat de rolul ei. Și observați că schimbarea apare imediat în fereastra “Listing”.

Executarea programului

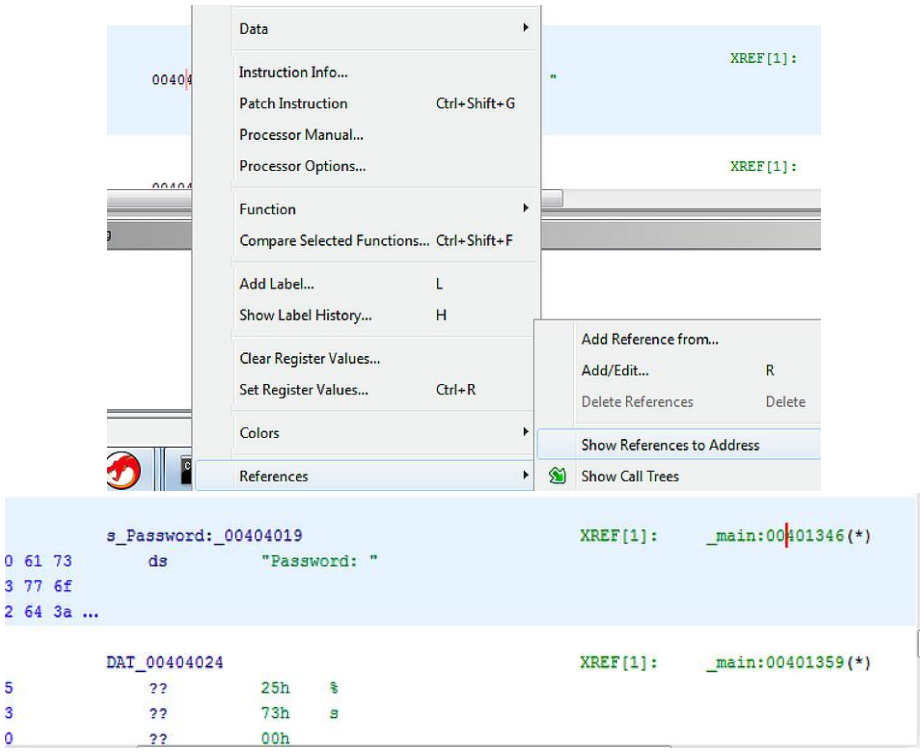
În cadrul execuției se observă că programul cere o parolă care evident atunci când îi introducem una la întâmplare va o va raporta ca invalidă.



Să trecem la procesare primară a acestui executabil prin inspecția șirurilor programului cu ajutorul secvenței “Window -> Defined Strings”. Se observă o porțiune de text afișat în linia de comandă. Practic reluăm analiza prezentată la curs. Să analizăm secțiunea de cod ASM care tratează parola. Dați un dublu click pe câmpul “Password” din zona “Defined Strings” și veți fi dușI automat la secțiunea în care respectivul cod este păstrat în executabil.

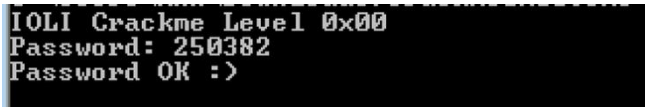


Dați click stânga pe adresă și selectați “References -> Show References to Address”. Apoi se poate da un click în respectiva zonă pentru a ne duce în zona de cos care conține rererirea la “Password”. O altă posibilitate este să se dea dublu click pe adresa aflată în dreapta sus a liniei care are o notă de tip XREF[1] note. Încercați să aflați ce secțiune de asamblare este responsabilă de verificarea parolei introdusă de utilizator. Eventual redenumiți unele variabile pentru a clarifica un pic codul.



Se observă că în referirea la “Password” există un apel de *scanf* pentru a prelua intrarea de la utilizator și apoi un apel de *strcmp*. Se observă că șirul dat de utilizator este salvat în EAX și păstrat în variabila locală “local_40”. Linia “250382” este de asemenea salvată în variabila locală “local_3c” și apoi amândouă sunt trimise către *strcmp*. Rezultatul comparației este apoi comparat cu zero și dacă este așa atunci se va afișa “Password OK :)” altfel se va afișam“Invalid Password!”. text.

Să executăm aplicația din nou dar de data asta cu presupusa parolă (“250382”). Se observă că de această dată parola a fost corectă.



Temă de laborator

1. **Se realizează testarea aplicației conform pașilor descriși în lucrarea de laborator.**
2. **Se reiau pașii asupra aplicației pay prezentată ca exemplu la curs.**

Tema pe acasă

Să se implementeze în C și apoi în CPP un program similar cu cele analizate apoi să se compileze în linux cu GCC și în C# nativ sau pe mono fără informații de depanare incluse și apoi să se reia procesul de analiză. Se va face un raport cu cod bază și capturi de ecran din timpul analizei (în zonele esențiale) pentru fiecare caz

Studenții care vor excela pot primi spre analiză malware real pentru a face un raport despre mostra primită - evident după ce semnează un agreement legal.