Anàlisi i Explotació de Vulnerabilitats en Sistemes de Validació de Números de Sèrie:

Un Estudi sobre Aplicacions Antigues.

Oscar Alvarez Barcons

Grau d'Enginyeria Informàtica
Seguretat informàtica

Tutor/a de TF
Gerard Farràs Ballabriga
Professor/a responsable de l'assignatura
Pau Perea Paños

06/2024

Universitat Oberta
de Catalunya

FITXA DEL TREBALL FINAL

Títol del treball:	*Anàlisi i Explotació de Vulnerabilitats en Sistemes de Validació de Números de Sèrie*
Nom de l'autor:	*Oscar Alvarez Barcons*
Nom del consultor/a:	*Gerard Farràs Ballabriga*
Nom del PRA:	*Pau Perea Paños*
Data de lliurament (mm/aaaa):	*06/2024*
Titulació o programa:	*Grau d'Enginyeria Informàtica*
Àrea del Treball Final:	*Seguretat Informàtica*
Idioma del treball:	*Català*
Paraules clau	*Enginyeria inversa, "cracking" i "keygenning"*

Resum del Treball

El treball de final de grau (TFG) se centra en l'anàlisi i l'explotació de vulnerabilitats en els sistemes de validació de números de sèrie per a registrar llicències d'aplicacions, especialment en aplicacions antigues on aquesta validació es fa en local. Aquestes aplicacions emmagatzemen les llicències dins del propi codi, o bé, hi apliquen algun algorisme per validar-les. El treball proposa estudiar els algorismes per tal de generar claus vàlides de manera fraudulenta o modificar el propi binari perquè n'accepti qualsevol altre de no vàlida.

Conseqüentment, es crearan diversos "crack me" per explorar i entendre les vulnerabilitats que permeten fer un ús fraudulent de les aplicacions sense adquirir-ne les llicències. Un cop estudiades aquestes vulnerabilitats, es proposa un "crack" i es programa un "keygen" per tal d'explotar-les. Finalment, s'apliquen aquestes tècniques en una aplicació real.

Abstract

The final degree project focuses on the analysis and exploitation of vulnerabilities in serial number validation systems when registering application licenses and, more especifically, in outdated applications where this validation is done locally. These applications store licenses within their own source code or apply some algorithm to validate them. The project aims at studying the different algorithms in order to fraudulently generate valid keys or modify the internal binary system so as to accept any other invalid key.

Consequently, different "crack me" will be created and observed in order to explore and understand the vulnerabilities that allow fraudulent use of the

applications without acquiring their licenses. Once these vulnerabilities are studied, a "crack" will be created and a "keygen" will be programmed to exploit them successfully. Finally, these techniques will be applied to a real application.

Índex

Llista de figures

1. Introducció

A la dècada dels vuitanta, va començar a sorgir la cultura “hacker”. Persones que havien adquirit alts coneixements en informàtica.

Igual que en la política i societat actuals, on hi ha un ventall bastant elevat i complex d’ideologies, en el món “hacker” ha succeït el mateix; des dels coneguts com a “White Hat”, els quals la seva moralitat és la d’utilitzar els seus coneixements per trobar falles de seguretat i reportar-les de manera honesta, així com tasques divulgatives per fer un món tecnològic més segur, fins als coneguts com “Black Hat”, els quals actuen de forma totalment contrària. Aquest últim grup del conjunt de “hackers” fa servir els seus coneixements per tal de trencar mecanismes de seguretat i sistemes, sovint, amb ànims de lucrar-se personalment o causar un perjudici a tercers.

Entre el blanc i el negre i continuant amb el símil de la vida real, hi ha infinitat de grisos: “Grey hat”, per exemple, alguns grups de “hacktivistes” que justifiquen algunes accions al marge de la llei, per tal d’aconseguir el que ells consideren un món millor.

El propòsit d’aquest document és estudiar alguns dels mètodes i tècniques que fan servir els “crackers”, aquest subgrup de “hackers” situats al marge de la llei, per tal de trencar els mecanismes de validació de llicències del programari per utilitzar-los sense adquirir una llicència vàlida. És a dir, de manera fraudulenta causant un perjudici econòmic a la persona o grup que ha creat el programa.

A tall d'exemple, segons la BSA (Business Software Alliance), en el seu últim informe de 2018[1], va publicar que al voltant d'un 37% del programari instal·lat en equips personals és il·legal. Com se’n pot extreure, aquest elevat nombre causa un perjudici econòmic a les empreses que els comercialitzen.

Autodesk[2], en aquest article del 7 d'agost de 2003, calcula que hi ha una pèrdua de tretze mil milions de dollars a causa de la pirateria a escala global. Quan fan aquesta afirmació engloben a totes les empreses que es dediquen a comercialitzar programari i no només a ells mateixos. Aquest fet fa notar que l'ús fraudulent de programari és una pràctica força estesa i no pas un fet aïllat.

Malgrat no donar una xifra estimada de les pèrdues, expliquen que aquestes pèrdues afecten directament el desenvolupament i millora del nou programari i a la pèrdua de llocs de treball. Per mitigar-ho, han creat un programa anomenat "Autodesk Piracy Protection" que busca educar i perseguir legalment als infractors el qual els hi ha permès recuperar 60 milions de dollars només a Nord-amèrica.

[1] https://gss.bsa.org/ (Data de consulta 08/04/2024)
[2] https://investors.autodesk.com/news-releases/news-release-details/autodesk-steps-anti-piracy-efforts-combat-13-billion-revenue (Data de consulta 25/05/2024)

D'altra banda, Microsoft, en el report anual d'aquest últim any 2023[3] tampoc ofereix estimacions de les pèrdues a causa de la pirateria tot i que la menciona, molt superficialment, com una causa de pèrdua d'ingressos. Malgrat això, és conegut que hi ha nombrosos activadors per aquest sistema operatiu i la seva solució ofimàtica, fet pel qual fa pensar que les pèrdues ascendeixen a bastants milers.

Microsoft, pels voltants del 2009, també va aplicar una campanya d'educació i conscienciació per evitar la pirateria. Aquesta campanya anomenada Windows Genuine Advantage la qual s'instal·lava a Windows XP en mode d'actualització, invalidava la majoria d'activadors i forçava a validar la llicència contra un servidor remot cada vegada que s'iniciava sessió o es volia instal·lar actualitzacions des de Windows Update. Malgrat els seus esforços, aquesta protecció va ser estudiada i vulnerada al cap de poc temps a causa del fet que els binaris es desaven en el mateix equip. Tal com es pretén demostrar en aquest document, aquesta pràctica deixa bastant vulnerables les proteccions que es volen implementar per una persona experimentada en la matèria.

Tot i no ser objectiu d'aquest document, la pirateria en jocs tampoc és menyspreable, segons Yves Guillemot, CEO de Ubisoft, explica en una entrevista a "GamesIndustry International"[4] que la pirateria en jocs físics per PC ascendeix entre el 93 i el 95%. Malgrat que no especifica xifres, considerant certes aquestes dades, és fàcil imaginar l'impacte econòmic que genera en aquest sector.

L'altra cara d'usar programari il·legal per registrar aplicacions i jocs de forma fraudulenta, és que no hi ha cap garantia de qualitat o, fins i tot, pot contenir "malware". Segons un estudi fet a octubre de 2023 per part Kaspersky[5], la prestigiosa marca que ofereix solucions de seguretat, calcula que la mitjana de programari piratejat a escala mundial ascendeix al 35%, sent els habitants d'Amèrica Llatina els qui destaquen amb un 66% del total del programari. També apunten que aquest fet provoca 2.274 deteccions d'infeccions per minut i ho atribueix, majorment, al programari il·legal.

Més enllà del perjudici personal o les pèrdues econòmiques de tercers, l'ús fraudulent de programari amb llicència, pot comportar sancions econòmiques importants. Segons apunta Auratech[6], un despatx especialitzat en protecció de dades i ciberseguretat entre d'altres, gràcies a una denúncia del BSA, el jutjat de lo mercantil número 2 d'Alacant va imposar una sanció de poc més de 453.000€ a dues empreses valencianes per utilitzar il·legalment el programari Siemens NX 9.0. Per més detalls, es pot consultar aquesta sentència al web de "Diario la ley"[7].

[3]https://www.microsoft.com/investor/reports/ar23/index.html (Data de consulta 25/05/2024)
[4] https://www.gamesindustry.biz/guillemot-as-many-pc-players-pay-for-f2p-as-boxed-product (Data de consulta 25/05/2024)
[5] https://latam.kaspersky.com/about/press-releases/2023_america-latina-registra-2274-intentos-de-infeccion-de-malware-por-minuto-siendo-la-pirateria-el-mayor-villano (Data de consulta 08/04/2024)
[6] https://auratechlegal.es/uso-de-software-pirata/ (Data de consulta 08/04/2024)
[7]https://diariolaley.laleynext.es/content/Documento.aspx?params=H4sIAAAAAAAAEABXLQQ5AMBBG4dt0Kd2gm9lZ2aoDDH5MgmlaFW6P5Uu-J9NEbW-tLV3lXFmbCzGJHnTJguPE3zI_jY7-CaCZtwTDQ9Itn-ijkI8ZJvACr93KEfTRvH-rFpzC_QKll2_tYAAAAA==WKE (Data de consulta 09/04/2024)

1.1. Context i justificació del Treball

El punt de partida d'aquest treball recau en la importància de la seguretat informàtica i la importància de desenvolupar aplicacions segures.

Malauradament, la seguretat és un aspecte descuidat per molts desenvolupadors de programari i maquinari.

Des que s'ha intentat protegir el programari, hi ha hagut delinqüents que han intentat i, majorment aconseguit, saltar les proteccions. Algunes vegades, causant greus perjudicis econòmics. És aquí on recau la rellevància d'aquest treball.

En el transcurs d'aquest, s'aplicaran algunes tècniques d'enginyeria inversa amb un "debugger" per tal d'estudiar el comportament de diverses aplicacions a partir de les instruccions en llenguatge assemblador. Un cop es té aquest coneixement, es pot crear un generador de claus vàlides o directament modificar el binari per tal que accepti claus no vàlides.

L'objectiu és que el lector pugui evitar aquests mètodes de validació vulnerables en aplicacions futures o millorar-ne les ja existents.

1.2. Objectius del Treball

L'objectiu d'aquest treball és estudiar i exposar alguns dels mètodes que es fan servir per trencar algunes proteccions del programari així com han anat millorant. En aquest document es veuran alguns aspectes com:

- Cracking.
- Keygening.
- Reversing de programari per tal d'analitzar-ne el funcionament sense tenir el codi font.
- Descompilar aplicacions.
- Anàlisis dels recursos en aplicacions.

1.3. Impacte en sostenibilitat, ètic-social i de diversitat

Pel que fa a aquest document se'n poden extreure les següents conclusions respecte a l'impacte en sostenibilitat, ètic-social i de diversitat.

En primer lloc, pel que fa a la sostenibilitat, com que es tracta d'un document formatiu que no requereix cap maquinari específic més enllà del mateix ordinador personal i tampoc implica la generació de cap material addicional, no té cap impacte negatiu en el medi ambient que sigui significatiu.

Altrament, aquest projecte encaixa amb algun dels ODS (Objectius de Desenvolupament Sostenible) que proposa Nacions Unides per l'agenda 2030:

- **ODS 8**: Treball decent i creixement econòmic:

Objectiu: Contractar professionals qualificats i experts en la matèria amb condicions laborals i salarials òptimes per fomentar el desenvolupament d'aplicacions cada cop més segures i la implementació de nous sistemes que dificultin al màxim el robatori de llicències, contribuirà al creixement econòmic sostenible.

Accions:

1. Contractar professionals qualificats i experts en seguretat informàtica per al desenvolupament de programari.
2. Oferir bones condicions laborals i salarials per retenir talent i fomentar la innovació en el desenvolupament de solucions de programari.
3. Implementar i/o millorar tecnologies de protecció de llicències per prevenir el robatori de propietat intel·lectual.
4. Fomentar programes de formació i conscienciació per a professionals de TI sobre els riscos de la pirateria de programari i les mesures per prevenir-la.

- **ODS 9**: Indústria, innovació i infraestructura:

Objectiu: Millorar la infraestructura tecnològica per oferir nous mecanismes de validació de llicències que augmentin la seguretat del programari i minimitzin el risc de pirateria.

Accions:

1. Invertir en la millora de la infraestructura de tecnologies de la informació per proporcionar entorns més segurs per a la validació de llicències de programari. Per exemple, implementació de servidors centralitzats per a la validació de llicències en lloc de dependre de validacions en l'entorn local menys segures.
2. Col·laborar amb empreses tecnològiques i experts en seguretat informàtica per a la investigació i el desenvolupament de noves solucions de validació de llicències segures.
3. Portar aplicacions d'escriptori al núvol.

En segon lloc, pel que fa al comportament ètic, és important considerar que aquest document ha estat elaborat únicament i exclusivament amb finalitats formatives: exposar vulnerabilitats amb l'objectiu de conscienciar els lectors sobre la seva existència per tal d'evitar que es cometin aquests errors en els seus dissenys actuals o futurs. Tanmateix, atès el contingut sensible, és important reconèixer la possibilitat que aquests coneixements puguin ser utilitzats per a fins il·lícits. L'ús responsable d'aquests coneixements sempre recau en el criteri del lector.

En tercer i últim lloc, la diversitat, el coneixement no entén de gèneres, nacionalitats ni col·lectius. Per tant, se'n pot extreure que és positiu per qualsevol persona que llegeixi aquest document i en faci ús, tant per aprendre com per millorar el seu programari.

1.4. Enfocament i mètode seguit

El treball està enfocat de manera que es crearan dos programes amb C molt senzills els quals implementen dues maneres de validar el número de sèrie que són vulnerables. L'objectiu d'aquestes és estudiar de la manera més fàcil possible els mecanismes o estratègies que es volen abordar en aquest document en lloc de donar una funcionalitat concreta. Un cop estudiats, es demostrarà com es poden comprometre aquests mecanismes amb l'ajuda d'un debugger i/o un desassemblador.

En aquestes demostracions es modificarà el binari per aconseguir saltar la protecció o bé obtenir una clau vàlida per poder registrar el producte sense comprar la llicència i es programarà un generador de claus vàlides.

Seguidament, s'aplicaran aquestes tècniques a una aplicació real vulnerable en la qual, a banda del que ja s'ha vist anteriorment, es mostraran algunes altres tècniques per arribar a la secció de codi a on es fa la validació i estudiar-ne el seu funcionament. També se'n programarà un generador de claus vàlides.

Finalment, es descompilarà un generador de claus ja existent utilitzat per registrar mapes en un sistema GPS d'un Peugeot 208 de manera fraudulenta. S'estudiarà el funcionament del codi i s'utilitzarà per buscar-hi similituds i diferències amb el codi del generador de claus obtingut en l'exemple anterior.

Cal afegir que, tot i que en els exemples proposats seran redactats de manera seqüencial, es farà èmfasis al fet que tot el procés consisteix en assaig i error sobre la base de teories que hom es pot fer. Malgrat que en aquest document s'utilitzi el codi font per fer l'explicació més comprensible, en un entorn real no serà d'aquesta manera i així es demostrarà en un exemple real.

Addicionalment a tot el comentat anteriorment, durant el transcurs d'aquest TFG, es va considerar oportú afegir un quart cas d'anàlisis on es demostrarà com es pot descompilar un generador de claus de tercers per entendre el mecanisme de validació aprofitant un treball ja fet per altres.

1.5. Planificació del Treball

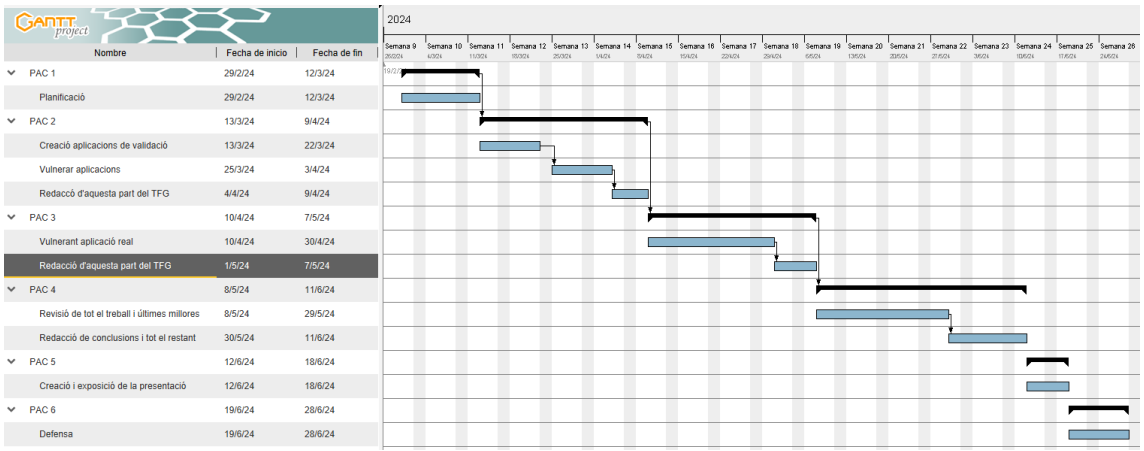

Figura 1: Planificació del Treball: Diagrama de Gantt amb Gantt Project.

- **PAC 1**: Es lliurarà el resum i la planificació del projecte.
- **PAC 2**: Es lliuraran els dos primers programes vulnerables juntament amb el seu binari modificat, així com també el generador de claus. Addicionalment, es reservarà temps per generar la memòria, encara que no de manera definitiva, anotant tot el procés detalladament. La realització dels darrers ajustos de format es durà a terme a la PAC 4 (Lliurament Definitiu). D'aquesta manera permetrà rebre "feedback" i, si escau, millorar el treball. Un cop millorat, es cuidarà tots els detalls.
- **PAC 3**: Es durà a terme l'anàlisi de l'aplicació real i es lliurarà el generador de claus. Es reservarà temps per generar la part de la memòria relativa a aquesta fase, tot i que, igual que en la PAC 2, no serà definitiva. A més, s'han destinat uns dies addicionals per implementar les millores suggerides en la PAC anterior. En cas de no existir cap suggeriment, aquest temps es dedicarà a millorar l'informe en preparació per al lliurament definitiu.
- **PAC 4**: Es faran les conclusions oportunes, es parlarà de treballs futurs, es cuidaran tots els aspectes de redacció i format i també s'implementaran, si s'escau, les millores proposades a la PAC anterior, per tal de garantir el millor lliurament final possible. En cas que, malauradament, hi hagi algun aspecte del treball endarrerit també hi hauria temps per acabar-ho.
  NOTA: A banda del ja mencionat, durant el transcurs d'aquest document es va decidir incorporar un quart exemple on s'analitza un generador de claus de tercers que permet activar mapes en automòbils Peugeot. Com que ja hi havia un temps destinats a millores per endarreriments, no suposarà cap modificació a la planificació.
- **PAC 5 i 6**: Presentació i defensa del treball.

Porto a terme aquest projecte mitjançant “sprints”. Els “sprints” són una metodologia popular en la gestió de projectes, especialment en entorns àgils com “Scrum”. Un sprint és un període de temps curt i fix durant el qual es fa un determinat treball.

Per fer-ho, divideixo l'objectiu final (tota la tasca prevista per a cada entrega) en parts diferenciades a les quals els estableixo una durada determinada. Aquesta metodologia em permet treballar en el projecte de manera més estructurada i saber en tot moment si estic treballant dins del termini. També permet certa flexibilitat per aplicar canvis si és necessari.

Aquesta metodologia contempla lliuraments periòdics dels diferents “sprints”. Aquests lliuraments són propis per a fer autoavaluacions, excepte cada lliurament planificat en el pla docent. En aquest cas, es lliurarà tot el previst en la planificació.

També es contemplen reunions periòdiques per detectar possibles mancances i àrees de millora. En aquest cas, es duran a terme durant el temps de treball per correu electrònic amb el tutor si es considera oportú, i una "reunió" final que correspon al “feedback” de cada lliurament per part del tutor.

El material utilitzar serà el següent:

- Un compilador de C:
  - GCC 11.4.0 instal·lat en un sistema Ubuntu 22.04.4 LTS[8].
  - Juntament amb el paquet de biblioteques mingw-w64 (x86_64-w64-mingw32-gcc (GCC) 10-win32 202201113) per compilar aplicacions per Windows des de Ubuntu.
- VirtualBox[9]: Només s’utilitzarà per instal·lar-hi un Ubuntu 22.04.4 LTS amb la finalitat de compilar els dos binaris pels dos primers exemples i els generadors de claus.
- Un debugger (x64dbg[10]).
- Un decompilador (IDA free[11]).
- HDD Low Level Format Tool[12] versió 4.40 en la seva versió sense instal·lador (Aplicació real a vulnerar).
- Un editor de recursos: Resource Hacker[13] a la versió 5.2.7.
- Un IDE (Net Beans[14]) a la versió 21.
- JRE (Java Runtime Environment) a la versió: 8 update 411.
- JDK (Java Development Kit) a la versió 22.
- Es mencionaran altres recursos de forma teòrica que poden ser útils.

[8] https://ubuntu.com/ (Data de consulta 08/04/2024)
[9] https://www.virtualbox.org/ (Data de consulta 08/04/2024)
[10] https://x64dbg.com/ (Data de consulta 08/04/2024)
[11] https://hex-rays.com/ida-free/ (Data de consulta 08/04/2024)
[12] https://hddguru.com/software/HDD-LLF-Low-Level-Format-Tool/ (Data de consulta 08/04/2024)
[13] https://www.angusj.com/resourcehacker/ (Data de consulta 08/04/2024)
[14] https://netbeans.apache.org/front/main/download/

1.6. Breu sumari de productes obtinguts

Els productes obtinguts durant la realització d'aquest document són:

- Dues aplicacions molt senzilles (sense funcionalitat), que se centraran en el procés de validació de la llicència. Cadascuna d'elles valida la llicència d'una manera diferent.
- Referent a les dues aplicacions anteriors. En una d'elles es modificarà el binari per tal d'aconseguir validar claus fraudulentes. Per tant, s'obtindrà un binari diferent de l'original. En l'altre, s'obtindrà un generador de claus vàlides.
- Durant l'estudi d'una aplicació real s'obtindrà un generador de claus vàlides.
- Finalment, s'obtindrà el codi font d'un generador de claus descompilat, creat per tercers, amb llenguatge Java.
- Un repositori a GitHub[15] amb els codis generats als dos primers exemples. La resta de codis no es farà públic, ja que es cometria un fet il·lícit en tractar-se d'aplicacions protegides per drets d'autor. No obstant això, s'ajunten juntament amb la memòria per la seva avaluació.

1.7. Breu descripció dels altres capítols de la memòria

Materials i mètodes:

- Eines que s'utilitzaran.
- Base teòrica en llenguatge assemblador necessària per explotar les vulnerabilitats.
- Estudi del primer codi en C. Veurem un exemple on existeix la contrasenya escrita al propi codi i la compara amb el que li entra l'usuari. Al punt 3: Resultats, s'aplica aquesta teoria a la pràctica:
    - Trobar la clau vàlida.
    - Modificar el binari perquè n'accepti qualsevol de no vàlida.
- Estudi del segon codi en C. Aquest codi aplica un algorisme per validar la clau.
    - Comprensió de l'algorisme i veure de forma teòrica com fer un generador de claus. La part empírica d'aquest punt també es veurà al punt 3: Resultats.
- Vulnerant una aplicació real (HDD Low Level Format Tool).
    - Tot el que cal saber abans de posar-se a mans a l'obra. Novament, tot el procés pràctic es veu al punt 3.
- Anàlisis d'un generador de claus per activar mapes en un cotxe Peugeot 208.
    - Tot el procés pràctic es veu al punt 3.

[15] https://github.com/oalvarezba/TFG-OALVAREZBA (Data de consulta 10/06/2024)

Resultats:

- Pas a pas de la vulneració del mecanisme de registre de la primera aplicació.
- Pas a pas de la vulneració del mecanisme de registre de la primera aplicació.
- Pas a pas de la vulneració del mecanisme de registre de l'aplicació real.
- Codi font descompilat del generador de claus per l'activació del mapes.

Conclusions i treballs futurs:

- Conclusions i anàlisis del que s'ha obtingut.
- Treball futur: Analitzar una aplicació que validi contra un servidor.

2. Materials i mètodes

En el món actual, on la informàtica exerceix un paper fonamental en gairebé tots els àmbits de les nostres vides, la seguretat de les aplicacions informàtiques s'ha convertit en una preocupació crítica. Des de les aplicacions bancàries fins als sistemes de control industrial i passant per les aplicacions d'escriptori, la integritat i la seguretat de les dades són aspectes fonamentals que cal protegir contra amenaces potencials.

Una de les àrees de major importància en la seguretat de les aplicacions informàtiques és la validació de les dades d'entrada. Les dades d'entrada incorrectes o introduïdes de manera maliciosa poden conduir a una sèrie de vulnerabilitats i atacs, que van des de la denegació de servei fins a l'execució de codi no autoritzat. Aquest document se centrarà en la importància de la validació dels números de sèrie.

Els números de sèrie són utilitzats en una àmplia gamma d'aplicacions per controlar l'accés i la distribució de programari. Aquests números actuen com una mena de "clau" que permet als usuaris accedir i usar determinades funcionalitats o recursos d'una aplicació. Per tant, la validació dels números de sèrie esdevé fonamental per assegurar que només els usuaris autoritzats tinguin accés al programari.

Mitjançant enginyeria inversa és possible explotar les vulnerabilitats en els mecanismes de validació dels números de sèrie per eludir les restriccions d'accés i fer servir el programari de manera no autoritzada.

Aquest document té com a objectiu principal explorar els mecanismes de seguretat en la validació dels números de sèrie en aplicacions informàtiques i fomentar la prevenció d'aquests. De la mateixa manera, vol conscienciar dels perills que comporta usar programari de tercers per registrar el programari de forma il·legal.

Per fer-ho, es mostraran un parell de codis escrits en llenguatge d'alt nivell C els quals presenten vulnerabilitats i s'explotaran:

En el primer dels codis, s'incorpora el número de sèrie al mateix codi, el qual es compararà amb el que s'escriu. És a dir, ho rebutjarà tot excepte la paraula clau en qüestió. Es veurà com es pot trobar aquesta clau mitjançant el codi en llenguatge assemblador: tant a la memòria durant l'execució del programa com mirant les cadenes de text del propi binari. A continuació, es modificarà el binari perquè el seu comportament sigui a la inversa. És a dir, que accepti qualsevol clau excepte la que el programador ha escrit al codi com a vàlida.

Aquest exemple està basat en una aplicació corporativa i no comercial que recull unes dades, les comprimeix amb una contrasenya que, igual que en l'exemple, està present en el codi i les envia a un servidor remot. Com a usuari d'aquesta aplicació, pot interessar conèixer quines dades s'estan transmetent més enllà de tenir fe cega en el fabricant.

En el segon dels codis, la funció d'avaluació s'ha millorat i ara segueix un algorisme. És a dir, hi ha diversos números de sèrie que poden ser vàlids sempre que compleixin una condició. L'objectiu d'aquest exemple és arribar a entendre aquest algorisme a partir del codi assemblador i crear un generador de claus vàlides.

Aquest exemple vol apropar-se, tot i que de manera senzilla, a com validen alguns programes comercials els quals és possible instal·lar el dia d'avui. Vol posar en evidència que no és suficient en dissenyar un algorisme en el codi, ja que, un cop s'aconsegueix esbrinar com actua, és molt fàcil generar claus que el validin.

Per acabar els casos pràctics amb llenguatge assemblador, s'analitzarà el codi del programa HDD Low Level Format Tool en la versió 4.40 i en la seva versió sense instal·lador. Aquest programari no és lliure, per tant, no se'n pot accedir al codi font, i, per tant, a diferència dels dos exemples anteriors, no es disposarà del codi en alt nivell. S'aplicaran tècniques que podria utilitzar un delinqüent per utilitzar el programari sense adquirir una llicència vàlida.

El propòsit de HDD Low Level Format Tool és eliminar totes les dades de diferents tipus de memòries a baix nivell. És un programa el qual en té propietat de HDDGURU i protegit, en data de creació d'aquest document (2024), per Copyright[16].

Aquest programa es pot fer servir de manera gratuïta amb una limitació de la velocitat de formatge a 50MB/s. Es pot eludir aquesta limitació adquirint una de les dues modalitats de llicència que ofereix: personal per un equip o comercial per tres o més en la qual s'ofereixen descomptes.

[16] https://es.wikipedia.org/wiki/Copyright (Data de consulta 08/04/2024)

Finalment, en aquest quart i últim cas, deixant el llenguatge assemblador de banda, s'analitzarà un generador de claus ja creat per tercers utilitzat per instal·lar els mapes de manera fraudulenta en un sistema GPS d'un Peugeot 208 que incorpora un ordinador SMEG. Com és evident, no se'n farà la prova en un dispositiu real, ja que s'incorreria en un fet il·lícit.

El generador de claus està fet en JAVA. Es descompilarà fent servir "JavaDecompilers"[17], un recurs en línia per aquesta finalitat, i s'analitzarà quines accions fa amb l'objectiu de descobrir noves maneres de validar llicències a banda de les ja vistes en casos anteriors, així com demostrar que els sistemes emportats, en principi menys accessibles que un PC amb Windows, també són vulnerables.

Es desconeix com s'ha arribat a obtenir el sistema de validació. Molt probablement ha estat obtingut per una persona o grup de persones expertes en electrònica i informàtica les quals han bolcat i analitzat el contingut de la memòria ROM de l'equip SMEG en un PC. Tanmateix, tot i que menys probable, també podria ser que en algun moment s'hagués filtrat el generador de claus que utilitza la mateixa casa per generar els codis que suposadament s'entreguen al comprador: ja sigui perquè s'ha accedit de forma no autoritzada als equips i s'ha robat aquesta informació, o bé un treballador descontent amb l'empresa que ha decidit publicar-la.

Al punt 3 d'aquest document, es detallaran de forma exhaustiva tots els processos descrits anteriorment. Per comprendre aquest procés, s'assumeix que el lector té coneixements de programació en els llenguatges C, JAVA i assemblador, així com també coneixements de compiladors i del funcionament de la CPU.

Finalment, cal recordar que tot el referent en aquest document és merament instructiu i que, en cap cas, s'ha de fer servir amb finalitats il·legals.

2.1. Debugging

El “debugging” és el procés d'identificar, analitzar i corregir errors en el disseny del programari. Els “debuggers” són eines que faciliten aquest procés permetent als desenvolupadors inspeccionar l'estat del programa en temps d'execució. Aquestes eines proporcionen funcionalitats com l'execució pas a pas i la visualització de l'estat de les variables entre altres, facilitant la detecció i correcció de problemes de lògica o comportaments inesperats.

En condicions normals, aquest procés es fa sobre el codi d’alt nivell, no obstant això, en els casos exposats en aquest document, es farà sobre el codi assemblador.

[17] http://www.javadecompilers.com/ (Data de consulta 25/05/2024)

2.2. Anàlisis amb assemblador

Un cop compilat el codi, si no es disposa del codi font original, és necessari analitzar-lo en llenguatge assemblador per comprendre exactament com actua un determinat programa.

El codi assemblador és la representació final del programa que la CPU pot entendre i executar directament. Cada instrucció es tradueix a binari seguint el conjunt d'instruccions proporcionat pel fabricant de cada processador, permetent que aquest les processi i les executi. Així, la persona que analitza aquest codi és capaç de comprendre el funcionament del programa en qüestió.

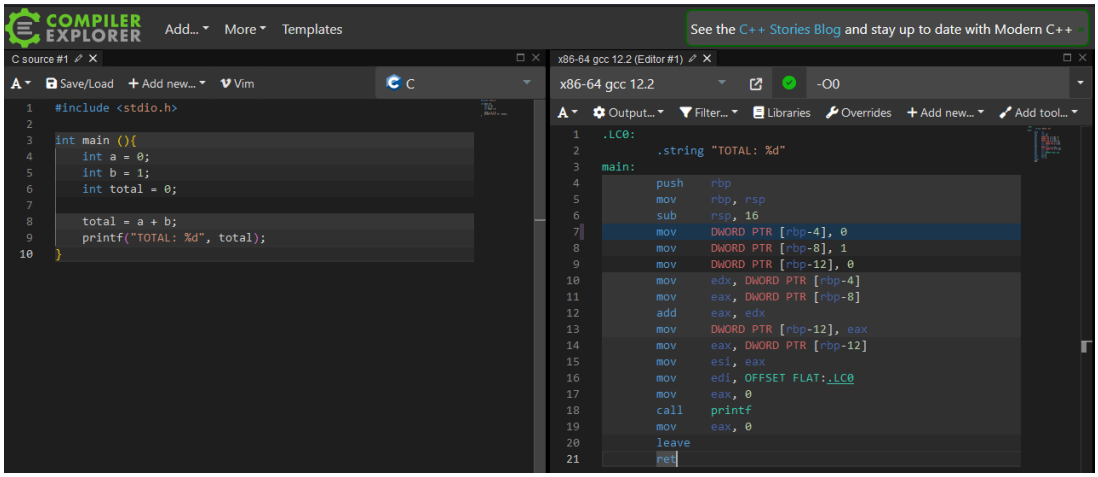

Figura 2: Anàlisis amb assemblador: Exemple introductori

Tal com es desprèn l'exemple anterior, unes poques línies en C, que tot programador experimentat pot llegir i entendre fàcilment, es converteixen en un munt d'instruccions bastant més costoses de llegir. Per tant, sempre que es disposi del codi font en un llenguatge d'alt nivell convindrà analitzar-lo en lloc d'optar per analitzar el codi en assemblador.

2.3. Comprensió del funcionament intern del programa

El "debugging" i l'anàlisi en llenguatge assemblador són complementaris i s'utilitzen conjuntament per comprendre el funcionament intern d'un programa del qual no disposem del codi font. Com s'ha pogut veure, és possible obtenir el codi en llenguatge assemblador a partir d'una aplicació ja compilada. Un cop es té accés a aquest codi, amb l'ajuda d'un "debugger" es pot seguir l'execució del binari en temps real.

En els casos que s'exposaran, en fer ús d'aquestes tècniques en conjunt, serà possible entendre els mecanismes de validació dels números de sèrie. Això inclourà aspectes com la longitud esperada del número de sèrie, el format previst de la informació de registre, l'algorisme usat per validar-lo, quines accions realitza la rutina de registre una vegada introduït un número de sèrie vàlid per generar persistència (com l'aplicació sap en el pròxim ús que disposem d'una llicència). En resum, totes aquelles accions que en puguin derivar del procés de registre.

2.4. Estructures en assemblador.

Abans de procedir amb els exemples proposats es veuran les estructures més comunes representades en aquest llenguatge amb la finalitat que, quan s'estudiï el codi, sigui més fàcil d'entendre.

Per mostrar aquests exemples s'utilitzarà el recurs web "Complier Explorer[18]". Aquest recurs permet escriure un codi en un llenguatge d'alt nivell i obtenir-ne el codi màquina equivalent emprant diversos compiladors amb algunes opcions de compilat.

Funció: L'inici d'una funció s'identifica perquè es comença desant a la pila l'estat actual d'alguns registres (`push rbp`, en l'exemple) per tal que durant l'execució de la funció, aquests registres es puguin modificar i, a la finalització d'aquesta, recuperar els valors originals (`pop rbp`) per continuar amb l'execució de la funció pare (`ret`).

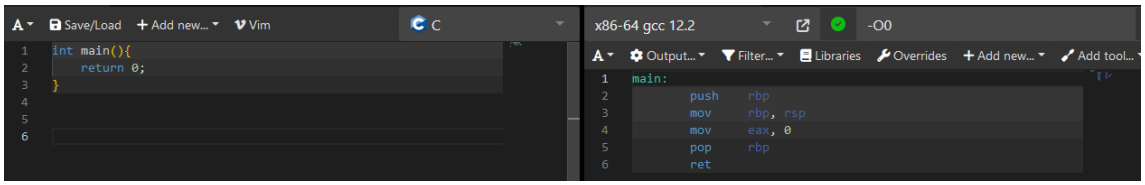

Figura 3: Estructures en assemblador: Funció

Condicional if: En aquesta estructura sempre es reprodueix el mateix patró:

- Es preparen les variables, sigui a la pila o en registres.
- Es produeix una comparació (instrucció `cmp` o `test`) la qual altera el bit del registre on es representa zf (zero flag) en funció de si els valors són idèntics o no.
- Es pren el salt o no (`jne` o bé `je`) en funció del valor de zf, de manera que s'executa el codi de dins el condicional if, o bé es continua amb l'execució del programa sense executar-lo.

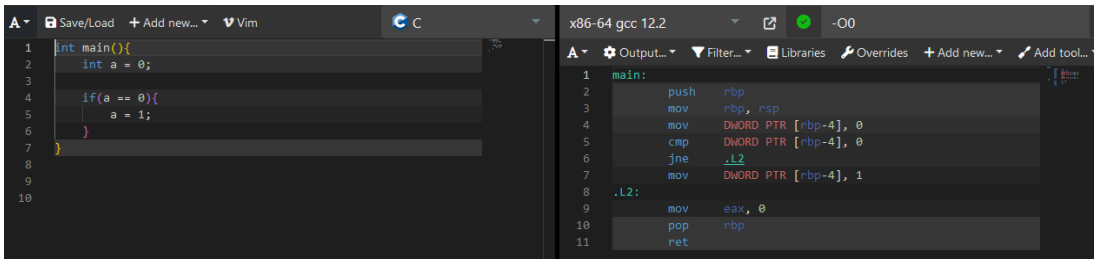

Figura 4: Estructures en assemblador: Condicional if

[18] https://godbolt.org/ (Data de consulta 08/04/2024)

Condicional if-else: El comportament és bastant similar al del condicional if:

- Es preparen les variables amb les quals es treballarà.
- Es fa la comparació alterant el bit del registre en el qual es representa zf.
- Es fa un salt a l'adreça on hi ha el codi que cal executar si no es compleix la condició o bé, en cas contrari, es continua l'execució seqüencial sense prendre el salt.
- Existeix la particularitat que, si es compleix la condició, un cop ha executat el codi pertinent, fa un salt sense condició (`jmp`) cap al final de la sentència if-else. Això és així perquè en assemblador s'executa de forma seqüencial i, si no s'altera el flux d'execució del codi, després d'executar el codi a mesura que es compleix la condició, també executaria el codi de l'alternativa.

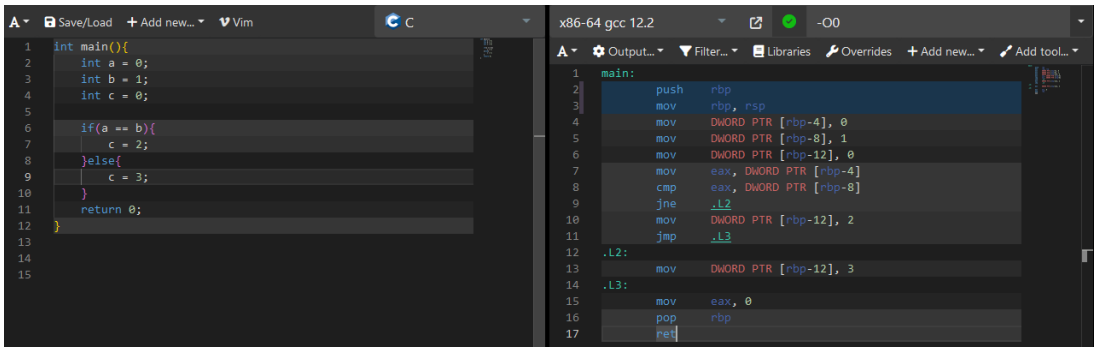

Figura 5: Estructures en assemblador: Condicional if-else

Condicional Switch: Segueixen el mateix mecanisme que els condicionals if-else: Comproven la primera condició. Si aquesta és certa, s'executa el cas i se salta al final del condicional (`jmp`). En cas contrari, se salta a l'adreça del següent cas on es torna a comparar si es compleix la condició. Aquest procés es repeteix fins a esgotar tots els casos programats. Si, així i tot, en cap cas ha estat avaluat com a cert, se salta a l'adreça del cas per defecte, s'executa el codi pertinent i es continua amb l'execució del programa.

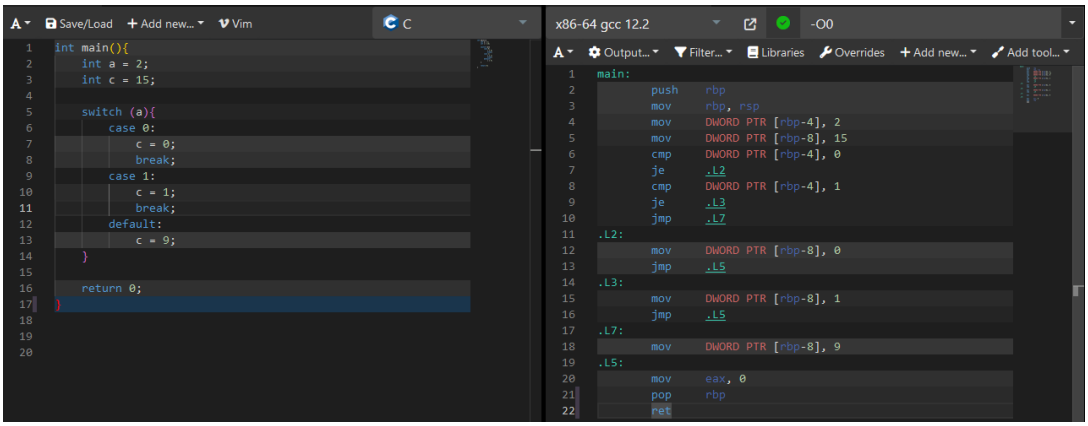

Figura 6: Estructures en assemblador: Condicional switch

Bucles: Prenen el següent aspecte:

- Es preparen els valors que s'utilitzaran (paràmetres de la funció) sigui emmagatzemant-los en registres o a la pila.
- Se salta a l'adreça on s'avalua la condició del bucle. Si aquesta condició es compleix, el programa continua amb l'execució. En cas contrari, salta a l'adreça on es troba el codi previst per ser executat dins del bucle.
- En acabar cada iteració del bucle, a l'última instrucció, el comptador s'incrementa en una unitat.
- És important assenyalar que, després d'actualitzar el comptador, no es produeix cap salt addicional, ja que la següent instrucció és la condició de bucle, i, com s'ha mencionat anteriorment, l'execució del programa és seqüencial.

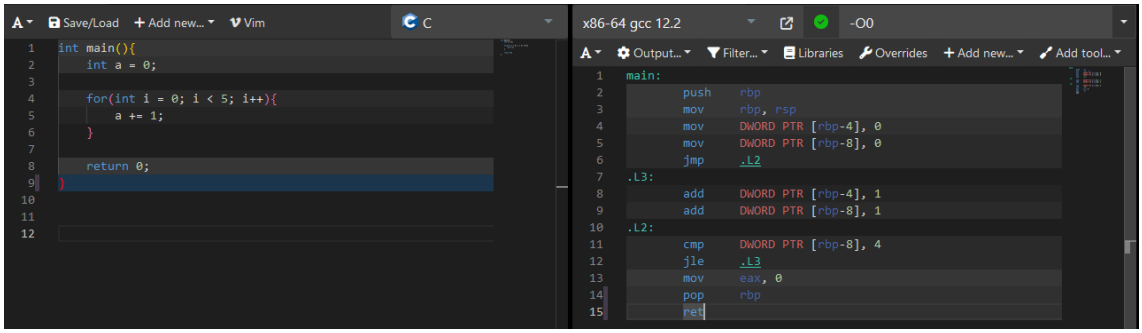

Figura 7: Estructures en assemblador: Bucle for

En aquest exemple s'ha fet ús d'un bucle for; no obstant això, el bucle while funciona de la mateixa manera. És important destacar que el bucle for és, simplement, una implementació del bucle while. Per tant, per evitar estendre més aquest document, es deixa a la voluntat del lector comprovar-ho.

Una altra variant és el bucle do-while; no obstant això, la mecànica és exactament la mateixa que en el bucle while, amb l'excepció que en aquest cas s'executa el codi per primer cop abans d'avaluar la condició.

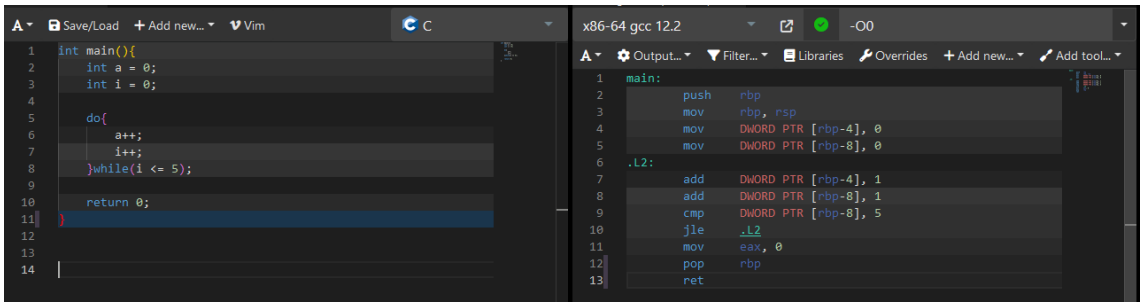

Figura 8: Estructures en assemblador: Bucle do-while

2.5. Entorn de Treball

A continuació es presenta l'entorn de treball que s'utilitzarà al laboratori:

Compilador: S'emprarà GCC 11.4.0 juntament amb el paquet de biblioteques mingw-w64 (x86_64-w64-mingw32-gcc (GCC) 10-win32 202201113), que permet compilar aplicacions de Windows en sistemes Linux. En aquest cas, s'utilitzarà Ubuntu 22.04.4 LTS executat en una màquina virtual mitjançant Virtual Box a la versió 7.0.14 r 161095 (Qt5.15.25).

Tots els executables es compilen amb les següents especificacions:

- x86_64-w64-mingw32-gcc -O0 -g programa.c -o programa.exe
    - -O0: Desactiva l'optimització del codi, ja que en aquests exemples es busca que l'anàlisi en codi assemblador sigui el més senzill i comprensible possible.
    - -g: S'inclou informació de depuració en el binari resultant.
    - Per defecte es complia per 64 bits.

En els dos exemples d'estudi es compilaran aplicacions de 64 bits amb l'objectiu d'analitzar aplicacions tant de 32 bits com 64bits ja que HDD Low Level Format Tool és un aplicació de 32 bits.

Debugger: Per aquesta finalitat s'usarà x64dbg versió snapshot_2024-03-27_00-26. Aquest paquet ofereix "debuggers" tant per 32 bits com per 64 bits.

A l'Annex 1 es pot consultar informació respecte a la interfície de l'aplicació i les funcionalitats que es faran servir.

Desensamblador: Per aquesta finalitat s'emprarà IDA Free a la versió 8.4.240320. Tot i que aquest programa ofereix una versió de pagament amb algunes funcionalitats afegides, amb la versió gratuïta serà suficient per desenvolupar aquest laboratori.

Editor de recursos: Per aquesta finalitat s'emprarà Resource Hacker a la versió 5.7.2. És un programa gratuït del creador Angus Johnson.

IDE: Per tal d'analitzar i provar el codi descompilat del generador de claus escrit amb JAVA, s'utilitzarà NetBeans a la versió 21 juntament amb el JDK 22 a la versió de 64 bits i el JRE 8 update 411 també de 64 bits.

3. Resultats

En aquesta primera part del document, es tractarà la part pràctica la qual es descriu de manera més teòrica al punt 2: Materials i Mètodes d'aquest mateix document.

Aquests exemples no pretenen ser una forma exhaustiva de resoldre aquest tipus de problemes, ja que el codi en llenguatge assemblador amb el qual treballem serà diferent per a cada aplicació. Inclús pot variar depenent de les opcions del compilador. És important que el lector entengui el concepte d'anàlisi del codi i la manera de modificar-lo, en lloc de considerar les següents instruccions de forma estricta com si es tractés d'un manual.

També és recomanable tenir al costat una llibreta o suport on poder anar anotant tot allò que pugui ser rellevant: valors fixos, adreces de memòria on el programa fa una acció que pot resultar interessant recordar-la en un futur o altres idees. Cal tenir en compte que analitzar un programa pot ser complex i pot portar fins i tot dies, si no s'anoten algunes coses és fàcil oblidar-les.

3.1. Cas pràctic 1

En primer lloc, en aquest primer exemple, es comença veient el codi font en C de l'aplicació amb la qual es treballarà. El codi es pot consultar el codi a l'Annex 2 així com també descarregar-lo a GitHub.

Es vol començar a treballar amb el primer binari Exemple_1.exe i, per tant, es carrega al binari al "debugger" (x64dbg). El primer que es fa avinent, tot i ser un codi petit, és que és força extens i que intentar entendre tot el codi línia per línia és una feina inviable i, aleshores, el lector es pot adonar que cal seguir altres estratègies que permetin anar directament a l'objectiu; si més no s'hi apropi força. En aquest cas per objectiu es considera aquella part del codi on es fa la validació.

Per aquest exemple i el següent, les següents reflexions poden semblar innecessàries perquè es disposa del codi font i, per tant, es podria arribar a l'objectiu amb relativa facilitat sense elles. No obstant això, l'objectiu d'aquest document és arribar a vulnerar un mecanisme de registre d'una aplicació de la qual no en disposem del codi font, tal com passarà al cas pràctic 3. Aleshores, tot i que el lector pot consultar el codi a l'Annex pertinent, s'actuarà com si no se'n disposés.

Dit això, abans de començar a treballar, el recomanable és executar l'aplicació i fer algunes proves del mecanisme de registre per tal de fer-se una idea de com pot estar funcionant. Recordem que, com que no es disposa del codi font, cal fer unes hipòtesis del funcionament i verificar-les al "debugger".

A l'executar Exemple_1.exe, es veu que es tracta d'una aplicació de consola. Només obrir-la demana la clau de registre per continuar i es queda esperant que l'usuari la introdueixi. Aquest fet ens permet afirmar que s'executa l'aplicació fins que requereix la interacció de l'usuari i, aleshores, el planificador de la CPU posa el procés en "standby". Aquest fet serà rellevant per posicionar-nos a prop de l'objectiu.

Seguidament, després de provar d'introduir la clau diverses vegades, es fa evident que sense la clau correcta no es pot continuar. En cada intent fallit, es mostra un missatge indicant que la clau no és correcte. A partir d'això, es pot inferir que hi ha una validació de la clau, tot i que aquest mecanisme encara no és conegut. L'objectiu és investigar com funciona aquesta validació. Un cop arribats a aquest punt, es pot suposar que quan s'introdueix una clau correcta, es mostra una notificació indicant-ho.

Malgrat que en aquest exemple aquest no es procedirà d'aquesta manera, en alguns casos pot ser útil utilitzar aquests missatges per acostar-se a l'objectiu, ja que si s'aconsegueix localitzar la part del codi on es mostra el missatge després d'introduir la clau incorrecta i s'analitzen les instruccions del voltant, és possible descobrir com es realitza la validació.

Ara si, després d'aquestes primeres hipòtesis es pot anar al "debugger" i posar-les en pràctica:

En primer lloc, s'executa l'aplicació fins que demana introduir la clau de registre. Efectivament, l'execució es posa en pausa a l'espera de què l'usuari la introdueixi.

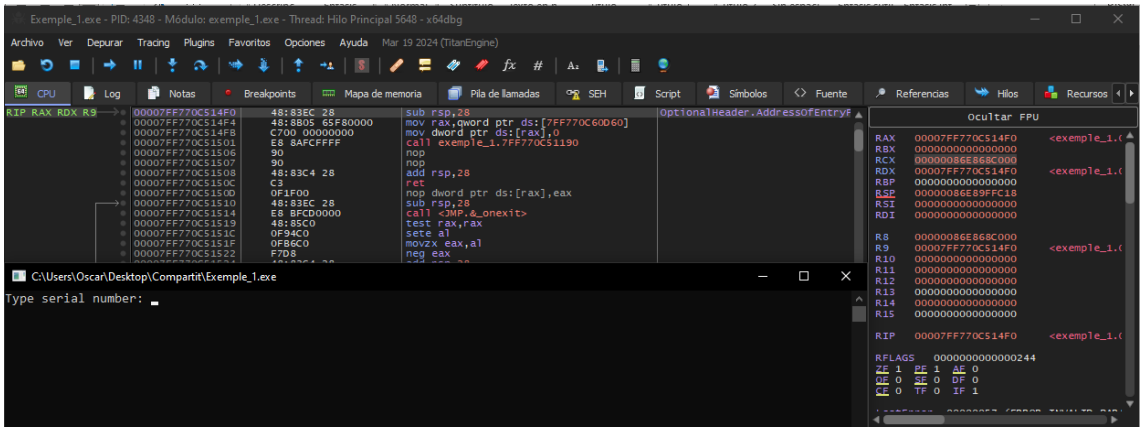

Figura 9: Cas pràctic 1: Primera execució de l'Exemple_1.exe a x64dbg

Arribats a aquest punt, es pot pensar que ens trobem bastant a prop de la secció del codi on es fa la validació. No ha de ser així necessàriament i sempre existeix l'opció d'executar pas a pas fins a arribar-hi. Com ja s'ha fet avinent, no existeix una solució única per tal d'assolir l'objectiu i, sovint, consisteix a prova i error.

En aquest cas, unes línies més avall, es pot veure que es crida a la funció de C "`strcmp`". D'aquí es pot treure una primera hipòtesi que s'està comparant dues cadenes de text i, que valida segons el resultat que retorna aquesta funció. És un bon punt per posar-hi un "breakpoint", introduir una clau de registre invàlida i continuar amb l'execució de l'aplicació.

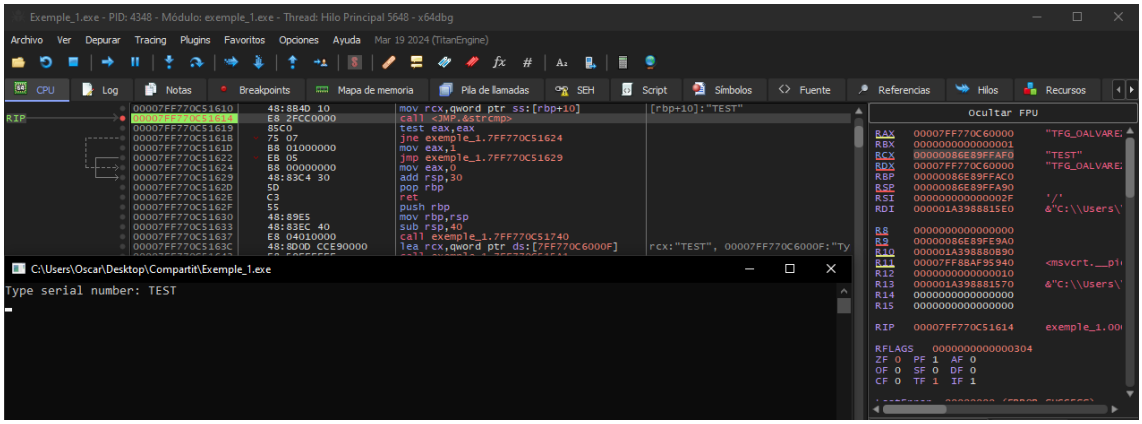

Figura 10: Cas pràctic 1: Call a la funció strcmp i clau als registres.

Ara, si s'observa l'estat dels registres es pot veure que, efectivament, s'està comparant la clau introduïda (TEST) amb TFG_OALVAREZBA. En aquest punt ja es coneix la clau que permet registrar l'aplicació. És fàcil comprovar-ho executant una altra instància de l'aplicació:

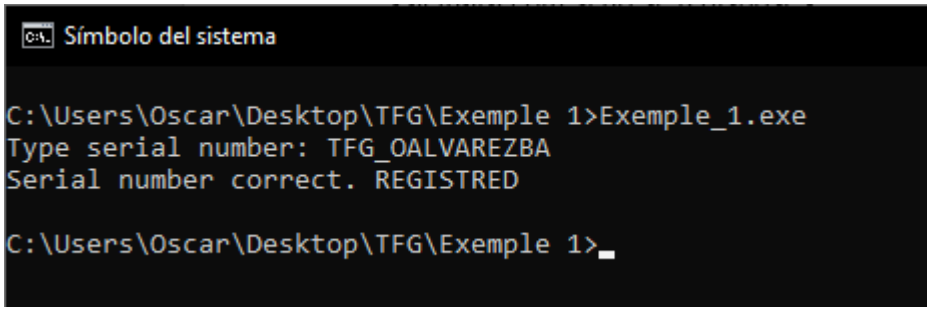

Figura 11: Cas pràctic 1: Aplicació registrada

A manera de detall, es pot veure que abans d'executar la funció "`strcmp`" es preparen els registres de la CPU. En aquest cas s'indiquen les respectives adreces on trobar les dues cadenes de text a comparar a la pila. És a dir, els paràmetres que utilitzarà la funció.

Arribats a aquest punt, ja s'ha assolit el primer pas: arribar a la part del codi on es fa la validació. Ara cal entendre el mecanisme de validació i, finalment, alterar-lo perquè accepti claus invàlides com a vàlides. Avançar fins a la següent instrucció ajuda a entendre que està succeint.

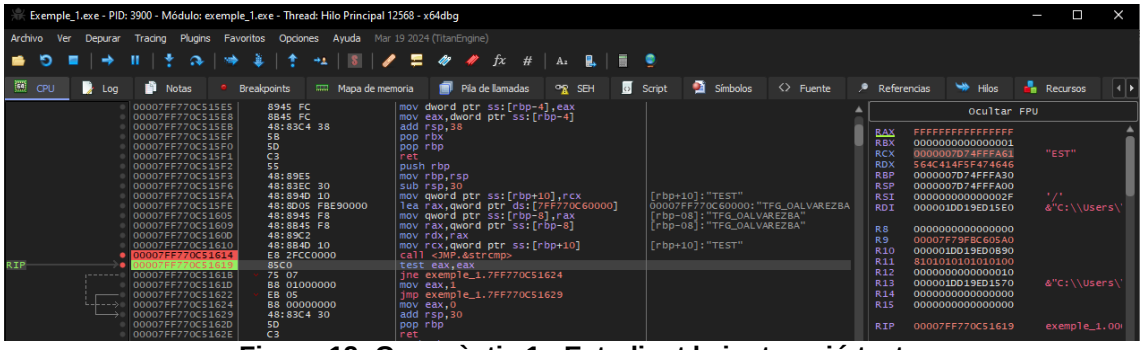

Figura 12: Cas pràctic 1: Estudiant la instrucció test

D'acord amb la documentació de Microsoft[19] la funció "`strcmp`" retorna el següent:

- Un valor < 0 si string1 és menor que string2.
- 0 si string1 és idèntica a string2.
- Un valor > 0 si string1 és major que string2.

I escriu el resultat al registre RAX.

A la següent instrucció (`test eax, eax`), avalua una AND lògica amb el valor del registre EAX.

És a dir, si hi ha un bit diferent de 0, al calcular `1 AND 1 = 1` i, per conseqüència, no s'activa la "flag" ZF (zero flag).

La "flag" ZF és decisiva en la següent instrucció de salt jne [@]. Aquesta instrucció salta a l'adreça indicada si el ZF val 0. I, per tant, es pren el salt cap a la instrucció `mov eax, 0` la qual mou un zero al registre RAX i finalitza la funció amb la instrucció de retorn `RET`. Cal recordar que aquest és el comportament en cas d'introduir un número de sèrie incorrecte (TEST, en l'exemple).

Un cop entès aquest procés podem fer-nos una idea de que hi ha una funció que actua de manera tal que:

- Si el valor del registre RAX és 0, retorna el missatge: No registrat.
- Si el valor del registre RAX és 1, retorna el missatge: Registrat.

[19] https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strcmp-wcscmp-mbscmp?view=msvc-170 (Data de consulta 08/04/2024)

Aleshores, el que interessa és modificar aquesta instrucció de salt perquè actuï a la inversa. Per fer-ho se substitueix la instrucció `JNE` per un `JE` amb l'opció d'ensamblar:

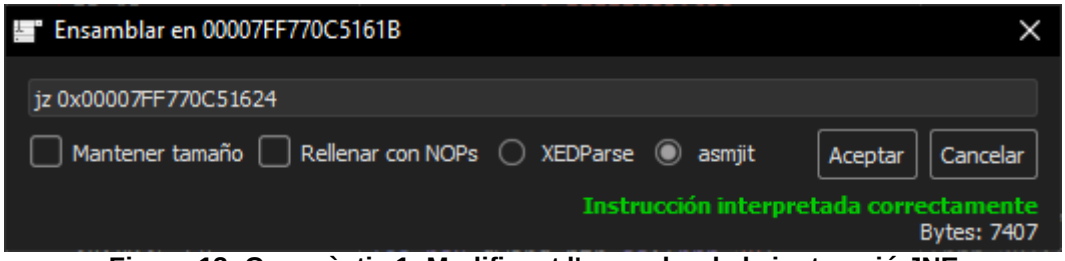

Figura 13: Cas pràctic 1: Modificant l'operador de la instrucció JNE

A continuació es pot observar com, efectivament, s'ha aplicat la modificació de la instrucció:

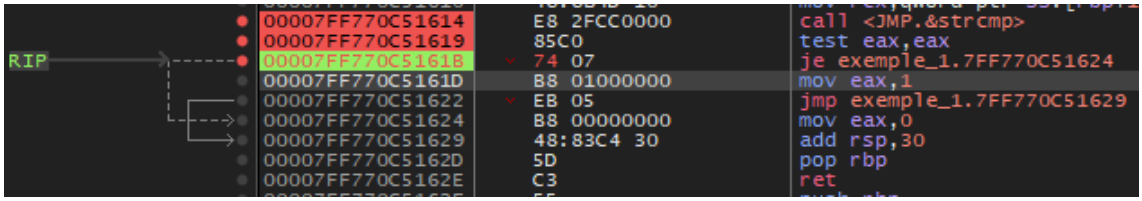

Figura 14: Cas pràctic 1: Instrucció JNE modificada per JE

Ara, es pot comprovar el funcionament del binari modificat continuant amb l'execució de l'aplicació. S'observa que, efectivament, l'aplicació registra amb un número de sèrie qualsevol.

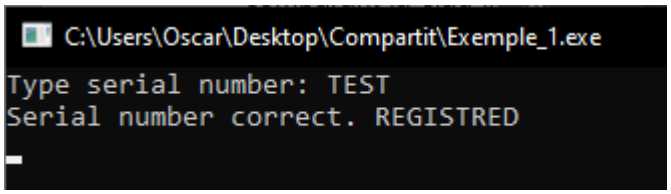

Figura 15: Cas pràctic 1: Registrant l'aplicació amb una clau qualsevol

Per concloure, només cal utilitzar l'opció per apedaçar el binari original amb la nova modificació per obtenir el que es coneix com a "crack".

Una altra manera d'aconseguir la clau TFG_OALVAREZBA és buscant les cadenes de text de l'executable. Una manera de fer-ho, és obrint l'executable amb un editor de text, bloc de notes de Windows, per exemple i, mitjançant alguna aplicació que permeti filtrar text amb expressions regulars, amb paciència també es pot arribar a obtenir:

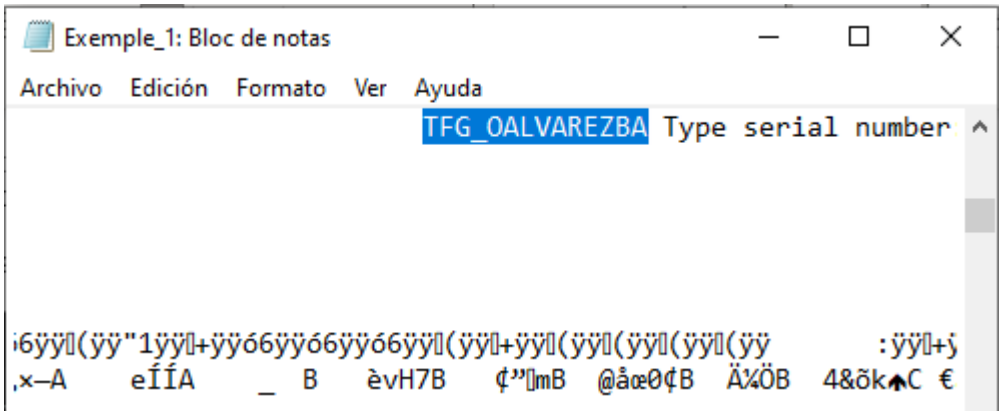

Figura 16: Cas pràctic 1: Aconseguint la clau vàlida amb un editor de text

Per complementar aquest exemple, s'enfoca el problema des d'una perspectiva diferent amb la finalitat d'oferir recursos al lector.

Cal recordar que s'aturava l'execució a l'espera que l'usuari introduís una clau de registre. En aquest moment, es podia haver enfocat el problema qüestionant-se què fa el programa amb què entra l'usuari.

Per exemple, es podria fer la hipòtesi que mira la mida d'una cadena de text per comparar una llargada en concret, que compara dues cadenes...

Partint d'aquesta idea, l'estratègia consisteix a buscar totes les crides entre mòduls que fa el programa, i veure si alguna pot coincidir, posar-hi un "breakpoint" i continuar amb l'execució i veure si s'ha arribat on s'esperava.

Aquest mètode pot resultar complicat degut a l'elevat nombre de crides que fa un programa, especialment si no es busca una teoria en concret. No obstant, si es va buscant un crida en concret, es poden filtrar els resultats pel nom de la crida i facilitar la feina.

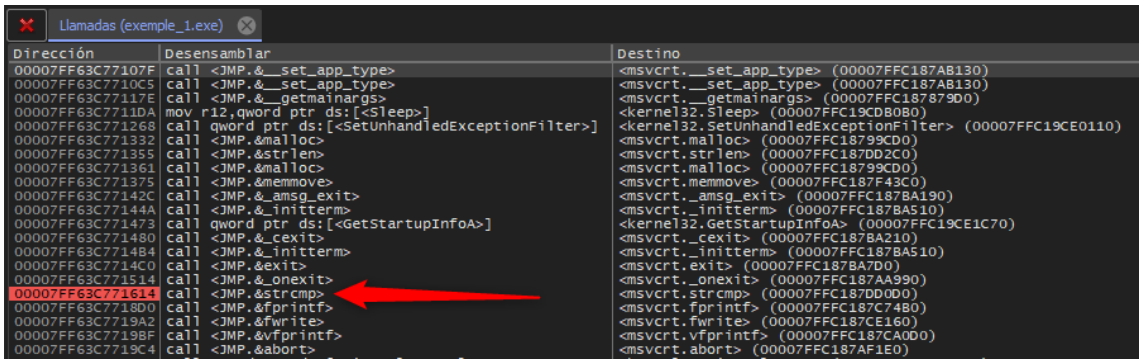

Figura 17: Cas pràctic 1: Abordant el problema des de les crides entre mòduls

3.2. Cas pràctic 2

En aquest segon cas, també es disposa del codi font perquè el lector l'utilitzi com a suport a l'Annex 3 i a GitHub, així com l'explicació detallada de les funcionalitats de x64dbg que s'utilitzen durant l'exemple a l'Annex 1.

Igual que al cas anterior, el primer que cal fer és provar l'aplicació per fer-se una idea de com pot estar construïda i treure algunes hipòtesis.

Després de fer les proves pertinents, es pot intuir que el mecanisme és molt similar al cas anterior. Aquest cop se seguirà l'estratègia ja comentada en el cas pràctic anterior d'aprofitar les cadenes de text que mostra el programa durant la seva execució.

En aquest cas pràctic, és interessant la cadena que mostra el programa en introduir una clau de llicència no vàlida: "Serial number incorrect.".

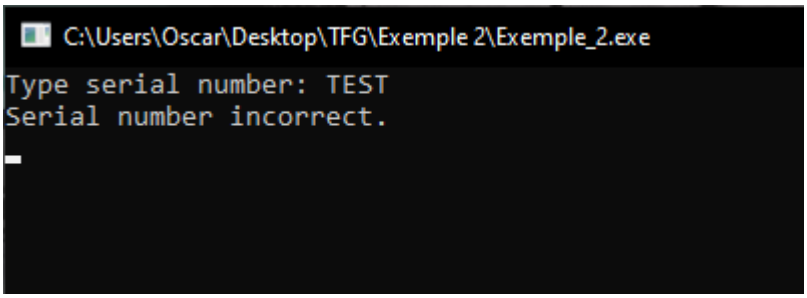

Figura 18: Cas pràctic 2: Primera prova amb una clau incorrecte

Amb aquesta informació anotada, ja es pot carregar l'executable d'aquest cas d'exemple al "debugger" i procedir a la seva execució fins que demani escriure la clau. Cal recordar que en aquest moment s'atura l'execució a l'espera que l'usuari escrigui una clau per llicenciar el programa.

Tal com s'ha mencionat anteriorment, x64dbg, permet buscar cadenes de text. En concret es vol trobar les cadenes de text del mòdul actual (Exemple_2.exe). Un cop localitzades, és convenient posar-hi un "breakpoint" a ambdues adreces amb la intenció de continuar l'execució del programa fins a arribar a un d'aquests dos punts i, un cop l'aplicació queda aturada, explorar el codi immediatament superior per veure si hi ha la possibilitat d'arribar al codi on es fa la validació. Amb sort, a la mateixa funció o a una de propera.

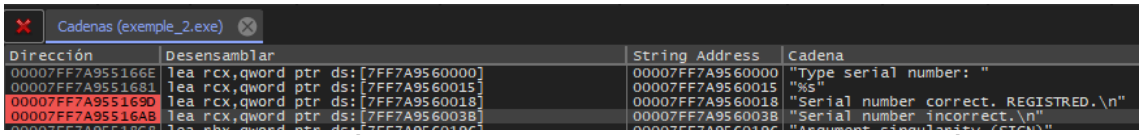

Figura 19: Cas pràctic 2: Cerca de les cadenes de text el propi mòdul

En el cas de l'exemple, poques instruccions més amunt, es troba el fragment del codi de la següent il·lustració. En aquest cas, la crida que es fa a la funció "`strlen`"[20] utilitzada per mesurar la llargada d'una cadena de text, ens pot fer sospitar que es pot estar mesurant la mida de la clau que s'ha introduït.

Altrament, unes línies més amunt, s'observa que es desa a la pila un valor un tant peculiar. No és mala idea anotar-lo.

Finalment, poques línies més amunt es veu com es desa a la pila l'estat actual dels registres `rpb` i `rbx` amb la intenció de poder restaurar-los després d'acabar aquesta funció.

Amb tots aquests indicis, és possible que estiguem davant de la funció encarregada de fer la validació. Aleshores, es pot considerar que és una bona idea posar un "breakpoint" a la instrucció `push rbp`.

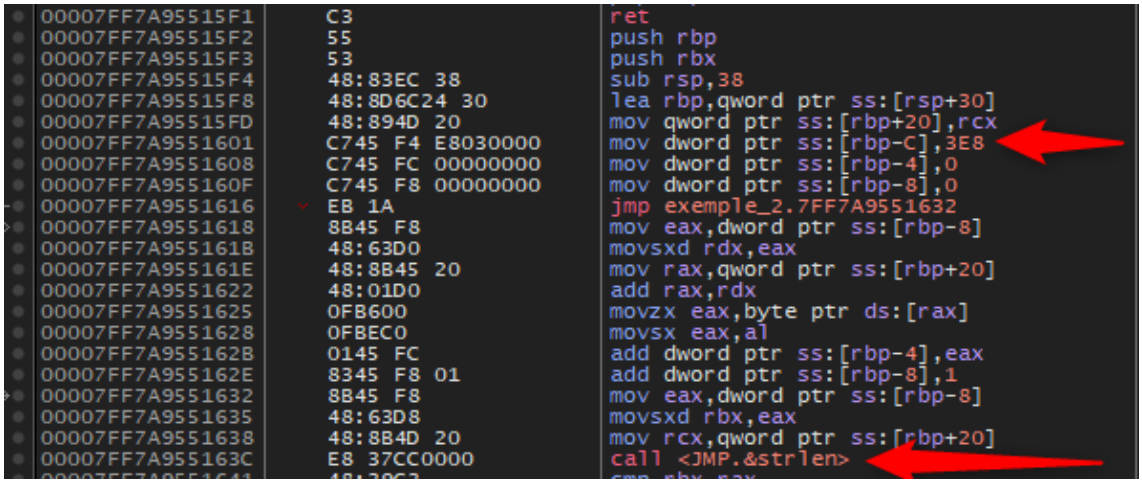

Figura 20: Cas pràctic 2: Alguns elements importants de la funció de validació

Un cop reiniciada l'execució del programa i, després d'introduir un altre cop un número de sèrie invàlid, es pot observar que, efectivament l'execució s'atura a on s'ha posat aquest últim punt d'interrupció.

[20]https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strlen-wcslen-mbslen-mbslen-l-mbstrlen-mbstrlen-l?view=msvc-170 (Data de consulta 08/04/2024)

Ara, s'utilitzarà el mode gràfic per veure els salts d'una manera més visual:

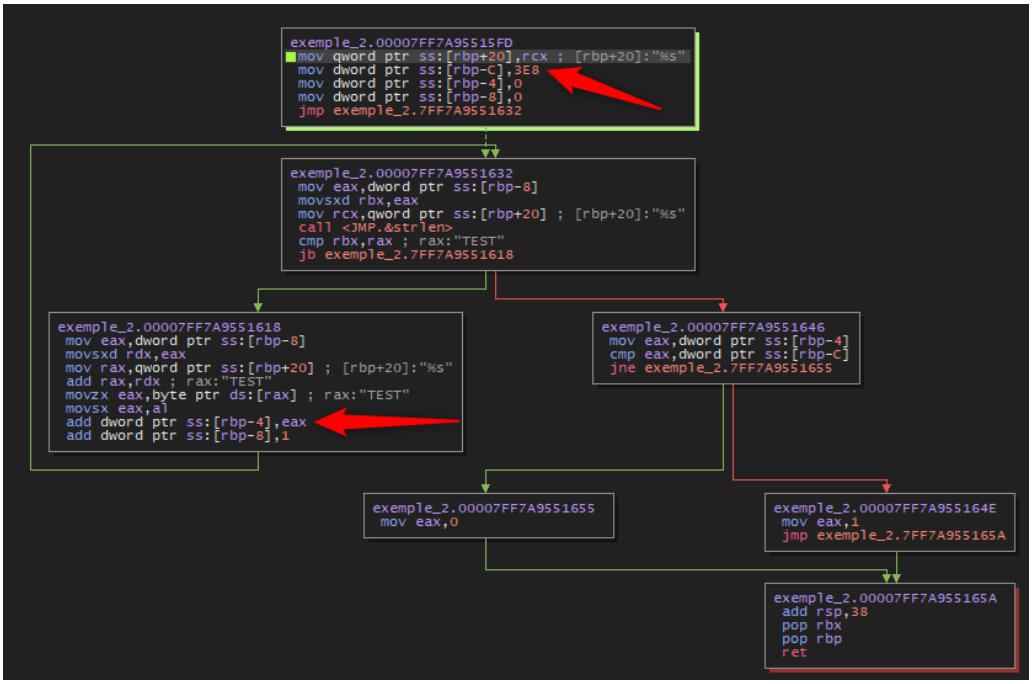

Figura 21: Cas pràctic 2: Visualització de la funció de registre en mode gràfic

Si s'analitza amb detall la branca de la dreta d'aquesta funció, el lector es pot adonar que segueix un mecanisme molt similar al cas anterior: es desa al registre RAX els valors 0 o 1 en funció de si s'ha aconseguit registrar correctament o no.

En aquest cas, la comparació que s'utilitza per decidir si la llicència és vàlida o no, es produeix comparant els 32 primers bits de més a la dreta del registre RAX amb els 32 bits de més a la dreta de la pila a l'adreça calculada de restar-li C al registre RBP.

Ara, doncs, cal veure com s'estableixen aquests dos valors:

El primer d'ells, el valor del registre RAX, s'estableix una instrucció més amunt d'aquesta comparació: `mov eax, dword ptr ss:[rbp-4]`. Aleshores, cal estudiar on s'estableix el valor de la pila a l'adreça `rbp-4`.

Mirant ara la branca de l'esquerra de la funció és fàcil veure que es produeix una suma a aquesta adreça de la pila (`add dword ptr ss:[rbp-4], eax`). Si es continua amb l'execució del programa, és fàcil adonar-se que és un bucle que es repeteix quatre vegades: un per cada lletra de la clau introduïda (TEST). Aleshores, el valor que es desa a l'adreça `rbp-4` de la pila és la suma ASCII de la clau introduïda:

TEST = 54+45+53+54 = 140 (Cal veure que la suma és en base 16 i no en base 10).

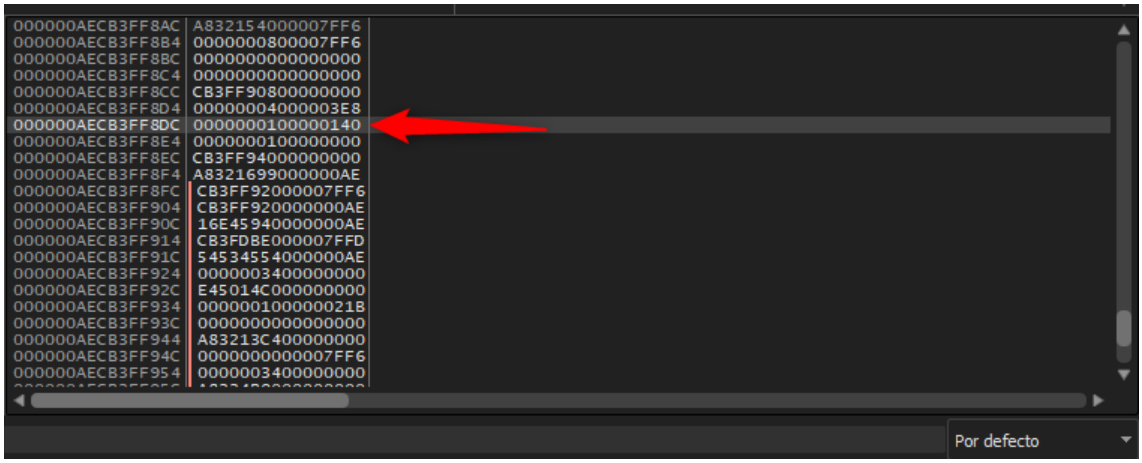

Figura 22: Cas pràctic 2: Resultat de la suma desat a la pila

Com a recurs addicional, i a tall d'exemple, en una aplicació real on el mecanisme és més complexa i treballant amb claus més llargues, aquesta refecció sobre el funcionament, pot no fer-se tan evident. Per aquesta finalitat, existeix un mecanisme per veure-ho a forma d'assaig i error.

Aquest mecanisme consisteix a establir un "breakpoint" a una instrucció qualsevol de dins el bucle i executar el mecanisme de registre de l'aplicació en qüestió fins a obtenir la notificació de clau incorrecte (o correcte, si s'introdueix una clau vàlida).

Aquest procés es fa amb la intenció que x64dbg incorpora una funcionalitat que va contant totes les vegades que el programa s'atura a un mateix "breakpoint". Si es coneixen dues claus invàlides de llargades diferents és fàcil adonar-se que realment, s'està produint alguna acció amb el número de sèrie.

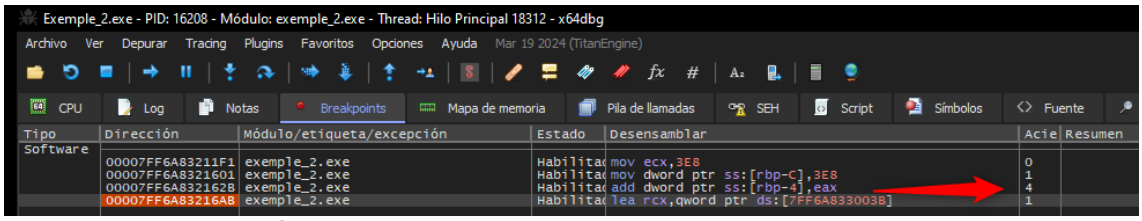

Figura 23: Cas pràctic 2: Comptador de les vegades que s'atura a un breackpoint

El segon d'ells, el valor que conté l'adreça de memòria de la pila `rbp-C`, es veu de forma directa que s'estableix al principi de la funció (`mov dword ptr ss:[rbp-C],3E8`).

Amb aquestes dades ja és possible explicar com funciona l'algorisme: Fa una suma ASCII de les lletres de la clau introduïda i, aquesta suma, ha de ser igual $3E8_{16}$.

Amb aquesta informació és senzill fer un generador de claus. Per fer-lo, en aquest cas, s'ha utilitzat compilador de C ja utilitzat en la creació dels dos primers exemples. El codi del generador de claus es pot descarregar de GitHub.

Cal tenir en compte que els valors vistos fins ara al “debugger” estan representats en base 16 (hexadecimal). Tot i que el "keygen" ja està pensat per aquest aspecte, a tall d'exemple i, per facilitar la comprensió, es faran les conversions i es presenta un exemple:

- $3E8_{16} = 1000_{10}$

Revisant la taula ASCII[21], es pot veure que la lletra "d" equival al valor 100 en base decimal.

Aleshores, la clau "ddddddddddd" (10 lletres "d") suma 1000_{10} i, per tant, valida.

Figura 24: Cas pràctic 2: Aplicació registrada

En la programació del generador de claus vàlides, s'ha tingut en consideració una longitud de 10 caràcters. Tot i que existeixen altres cadenes de text més llargues o més curtes que compleixen aquest criteri, també s'han tingut en compte únicament caràcters de l'alfabet (A-Z i a-z). Podria haver-se considerat l'addició de nombres, per exemple. No obstant això, com que la funció de validació no ho limita, no hi ha necessitat d'afegir-los.

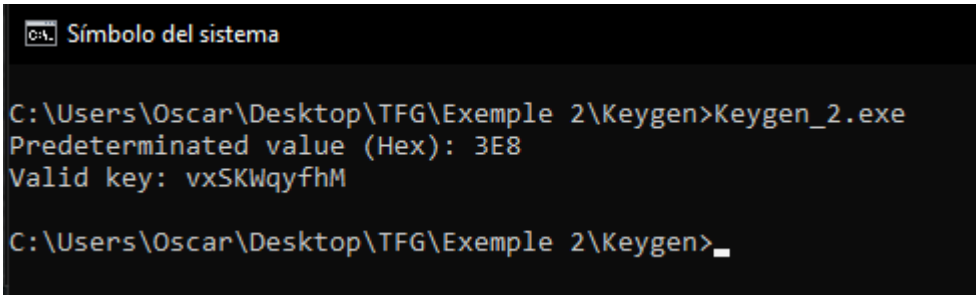

Figura 25: Cas pràctic 2: Generador de claus

[21] https://ca.wikipedia.org/wiki/ASCII (Data de consulta 08/04/2024)

Efectivament, el valor calculat, passa el mecanisme de validació i, per conseqüència, l'aplicació es registra:

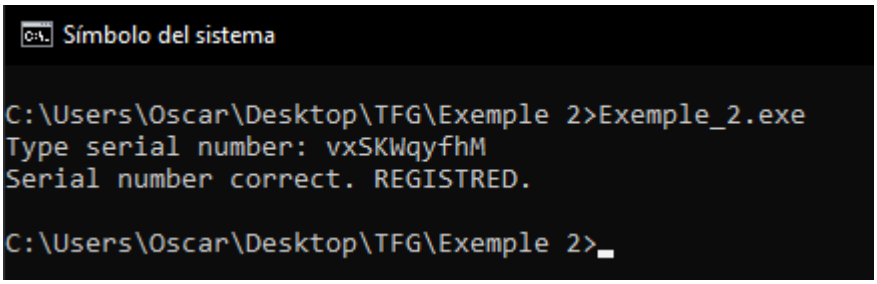

Figura 26: Cas pràctic 2: Introduint la clau generada

Fora d'aquest exemple i tenint en compte que, en aquest últim cas, s'ha vist el mètode de cercar cadenes de text per apropar-se a la part del codi on es produeix la validació de la llicència. El lector es pot preguntar: I si les cadenes de text no es troben directament al codi?

Es podria donar el cas que l'aplicació permeti seleccionar més d'un idioma per la interfície. Aleshores, cal estudiar el mecanisme amb què ho fa. Cal recordar que cada cas és particular i no hi ha una manera única de fer-ho. No obstant això, en aquest document s'exposa un mecanisme que pot ser vàlid en alguns casos.

L'estratègia és utilitzar un editor de recursos, carregar l'executable o llibreria corresponent i veure si hi ha una taula de cadenes de text amb la intenció de veure si és possible relacionar la cadena de text amb algun índex en alguna base de dades que pugui contenir l'aplicació. Vegem-ho amb un exemple:

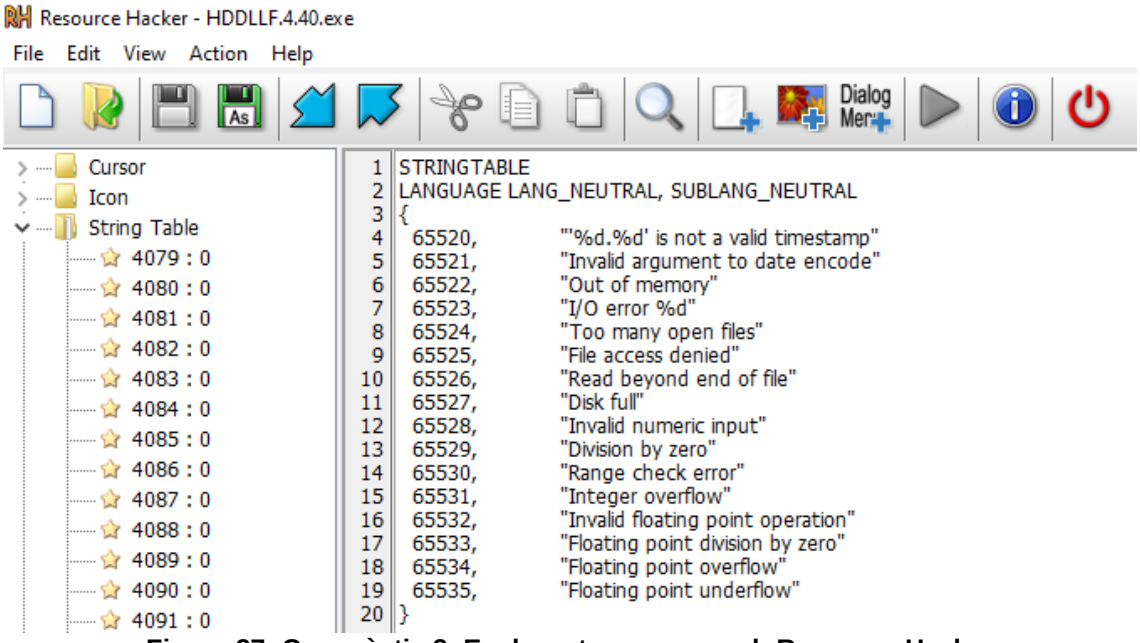

Figura 27: Cas pràctic 2: Explorant recursos amb Resource Hacker

En el cas de HDD Low Level Forma Tool existeixen un seguit de “strings”. Per exemple si es volgués veure on mostra el missatge “Disk Full”, caldria buscar al codi on utilitza l’índex 65527_{10} ($FFF7_{16}$). Per fer-ho cal obrir x64dbg i cercar constants amb aquest valor:

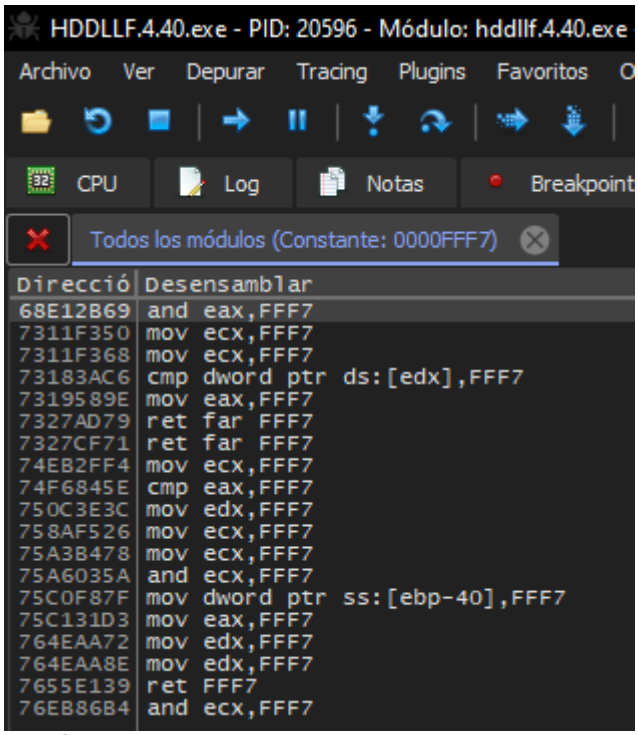

Figura 28: Cas pràctic 2: Buscant la constant de la cadena de text a x64dbg

El propi “debugger” ja fa el canvi de base i mostra totes les aparicions d’aquesta constant. Un cop trobades totes les instruccions que fan servir aquesta constant, cal destriar el gra de la palla. Podria ser que aquesta constant es fes servir, de casualitat, per alguna altra acció diferent.

Per fer-ho, es pot establir un punt d’interrupció a cadascuna de les aparicions i provocar que el programa mostri aquest missatge per tal que s’aturi a aquest punt. Es deixa en mans del lector experimentar amb aquesta opció.

3.3. Cas pràctic 3

En aquest tercer, tal com ja s'ha fet avinent, es deixaran els exemples de laboratori i s'estudiarà una aplicació real: HDD Low Level Forma Tool. En aquest cas no es disposa del codi font ni cap mena d'ajuda que faciliti la feina. El primer que cal fer, tal com s'ha fet en els dos exemples anteriors, és obtenir el màxim d'informació possible per basar les hipòtesis. Cal recordar que com més accions es puguin afirmar sense llegir codi en assemblador, millor ja que analitzar codi en assemblador és molt costós quant a temps.

En un principi, el primer que es pot fer és accedir al seu lloc web i veure quins tipus de llicències es poden adquirir:

License terms:

- **Free for personal/home use** (speed is capped at 180 GB per hour which is 50 MB/s)
- Just $3.30 for personal/home use (no speed limit): Order Personal license
- $27.00 per seat for commercial or professional use: Order Commercial license
- Updates are free for life

Figura 29: Cas pràctic 3: Opcions de llicenciament

D'aquí es pot concloure que el registre es du a terme per cada usuari. Conèixer això serà útil més endavant per facilitar la tasca d'entendre el mecanisme de registre. És a dir, un cop introduïda una llicència vàlida, a on es desa la informació pertinent per recordar-ho en properes execucions. Es pot pensar que ho farà a:

- A la carpeta del mateix usuari de Windows.
- Al registre de Windows a HKEY_CURRENT_USER.

Per aquest motiu, abans de començar l'anàlisi, és bona idea exportar una còpia d'aquesta carpeta del registre de Windows amb la intenció de poder-la comparar posteriorment i veure si hi ha modificacions després de registrar l'aplicació.

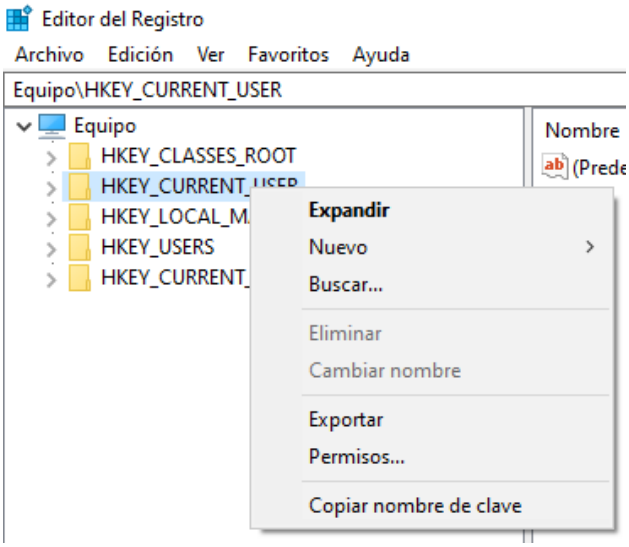

Figura 30: Cas pràctic 3: Exportació del registre

També es pot deduir que, d'alguna manera, quan es faci l'anàlisi, el mecanisme de validació distingirà entre aquestes dues modalitats de llicenciament.

Si s'intenta adquirir una llicència d'ús personal, se'ns redirigeix a una passarel·la de pagament on es demana introduir una adreça de correu electrònic. Això fa pensar que, un cop adquirida, el fabricant el respon adjuntant-hi una clau de registre. És necessari recordar que, malgrat aquest document mostri com generar claus fraudulentes, si es vol utilitzar el programa, aquesta és la única manera de fer-ho. Del contrari s'està cometent un frau.

De moment això és tot el que es pot obtenir del web per aquesta aplicació. A tall d'exemple, en altres aplicacions es pot aconseguir informació diferent com per exemple si és necessari generar un "token" al mateix equip i enviar-lo perquè el fabricant retorni el material necessari per al registre, instal·lar alguna aplicació addicional per fer efectiu el registre... Sovint, el fabricant ofereix instruccions al respecte on és possible obtenir informació sobre el funcionament del mecanisme de registre en qüestió.

En una segona fase, es pot procedir a executar el programa i a fer proves amb el mecanisme de registre. En primer lloc, es mostren uns termes d'ús de l'aplicació. Si s'accepten, es tanca l'aplicació i es torna a obrir s'observa que ja no es mostra més aquest missatge. El lector pot pensar que si es coneix on es desa aquesta acció, probablement serà al mateix lloc on es desi la informació de registre.

Tal com s'ha comentat, una opció és que ho faci a la carpeta de l'usuari local. Una estratègia és fer una cerca avançada per data de modificació de l'arxiu (com més acotada millor per tal d'obtenir el menor nombre de resultats possible) a la carpeta de l'usuari en qüestió:

- fecha:>=16/04/2024 17:58:00 fecha:<=16/04/2024 18:02:00

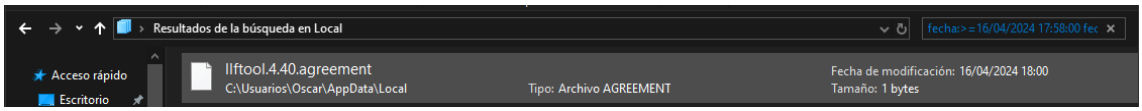

Figura 31: Cas pràctic 3: Cerca per data de modificació

Efectivament, el programa desa un arxiu anomenat llftool.4.40.agreement que es pot obrir amb un editor de text i que conté un 1 al seu interior, el qual es llegeix en iniciar l'aplicació.

Tot i que no es pot afirmar, es pot intuir que el mecanisme de registre de la clau funciona de la mateixa manera. Pot ser una bona idea anotar aquesta informació per recordar-la pròximament.

En cas que la cerca no sigui satisfactòria, es pot optar per exportar una còpia del registre i comparar-la amb la que ja s'ha fet en un inici amb l'ajuda d'alguna aplicació. Tot i que no és competència d'aquest document, se suggereix l'ús de RegistryChangesView v1.30 de NirSoft[22].

Un cop amb aquesta informació al poder es procedeix a fer unes proves amb el formulari de registre:

- Clau en blanc.
- Clau amb dos caràcters.
- 100% caràcters.
- 100% nombres.

Efectivament, no dona pistes com: La clau no pot ser buida, la clau ha de contenir X dígits (on X és un nombre), la clau no pot contenir nombres: ha de tenir X format (on X representa un format)... Sempre s'obté un mateix missatge indicant que la clau és incorrecte:

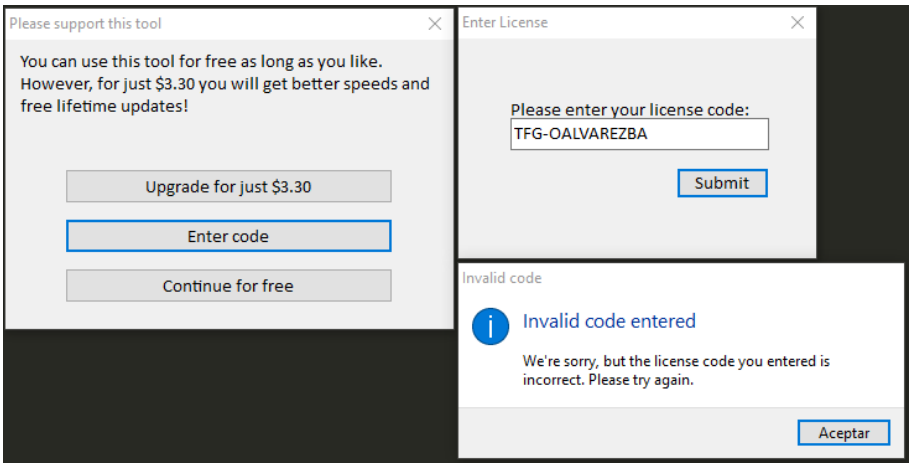

Figura 32: Introduint una clau de registre incorrecte

Aleshores no serà possible obtenir gaire més informació per aquí. No obstant això, sí que és possible observar uns fets:

- Genera un quadre d'alerta per informar que el codi és invàlid. Aquesta acció pot ser útil si es farà servir un explorador de recursos. En aquest cas s'utilitzarà Resource Hacker.
- En ser una aplicació que no permet múltiples idiomes i, que està formada per un sol arxiu, ens pot fer pensar que és possible trobar aquesta cadena de text al mateix codi. És una bona idea anotar-la per un futur.

[22] https://www.nirsoft.net/utils/registry_changes_view.html (Data de consulta 20/04/2024)

Una última prova que es pot fer per verificar que realment valida en local i no fa cap acció en un servidor remot és intentar validar mentre s'executa un analitzador de tràfic de xarxa: WireShark[23], per exemple. Ja s'avança que aquesta aplicació valida en local i, per tant, no és objectiu d'aquest document fer aquesta prova. Queda a voluntat del lector.

Ara ja s'ha recollit tota la informació que es podia obtenir com a usuari i és possible aproximar una idea de com funciona el mecanisme de registre:

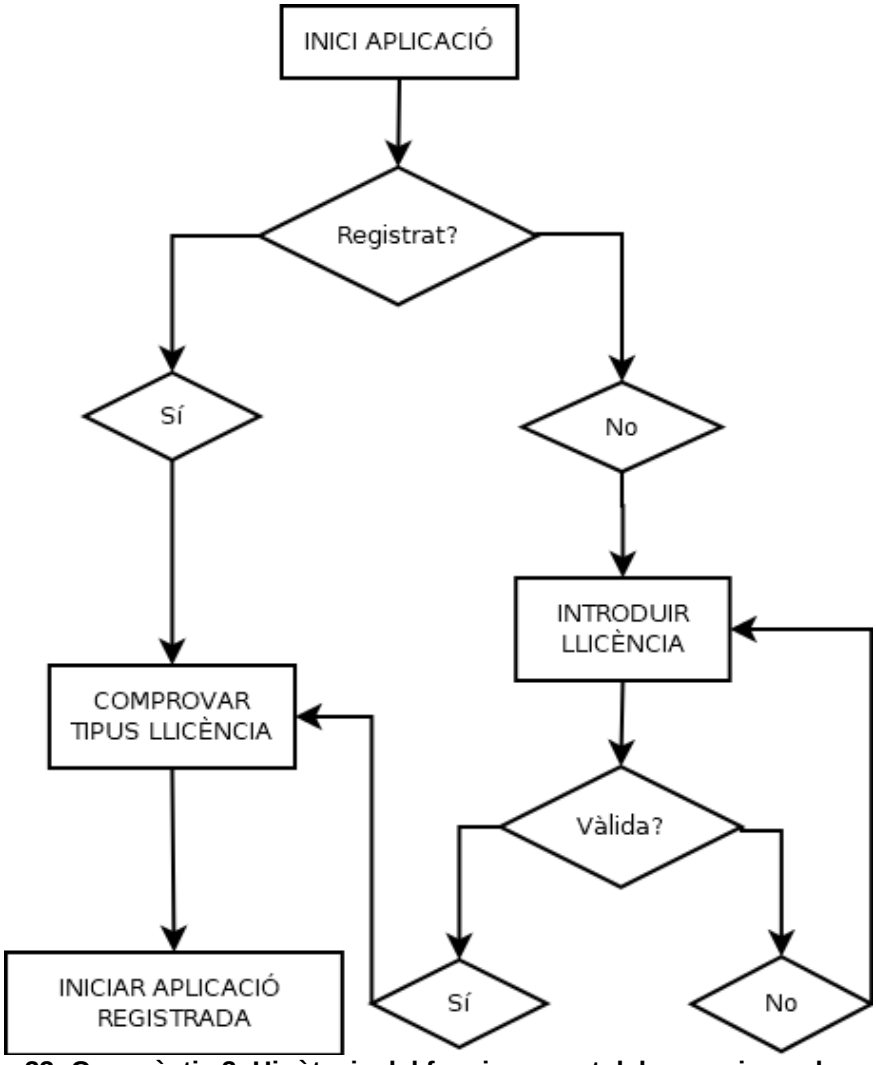

Figura 33: Cas pràctic 3: Hipòtesis del funcionament del mecanisme de registre

[23] https://www.wireshark.org/ (Data de consulta 20/04/2024)

El següent pas serà obrir l'aplicació amb Resource Hacker per tal de buscar dins dels recursos amb la intenció de trobar cadenes de text, noms de funcions, noms d'elements, etcètera, que permetin aproximar a la regió del codi on es produeix el registre:

Tal com es pot observar en la següent imatge, es troba un recurs anomenat "String Table". Si el lector explora aquesta carpeta, veurà que hi ha diversos textos que es poden mostrar durant l'ús de l'aplicació. No obstant això, no apareix el text vist anteriorment on s'informa que la llicència introduïda no és vàlida. Per tant, l'estratègia de posicionar-se al lloc a partir de la cadena de text que indica que la clau és incorrecte és fallida.

Malgrat això, si s'examina la carpeta RCData, s'hi pot observar un recurs força interessant: TFMENTERLICENSE. En seleccionar-lo es pot observar que correspon amb el diàleg amb el qual s'han estat fent proves anteriorment i també, que es desencadena l'acció després de prémer "Button1" (Submit). És una bona idea anotar aquesta informació.

A tall informatiu al recurs RCData (Resource Compiler Data), s'hi troben elements que el programa necessita per mostrar de forma dinàmica durant la seva execució.

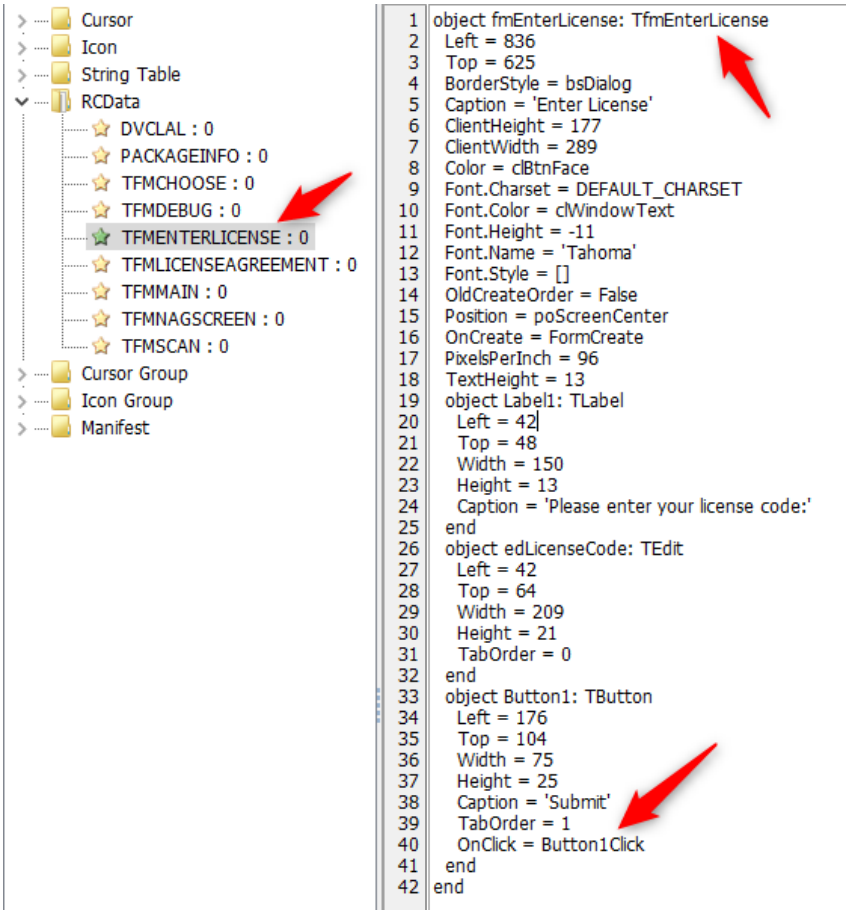

Figura 34: Cas pràctic 3: Informació del formulari de registre a RCDATA

Amb aquesta informació es pot accedir al descompilador. Com s'ha esmentat, s'utilitza l'IDA32 Free. Un cop s'hagi descompilat l'aplicació, aquesta apunta automàticament al Punt d'Entrada (Entry Point); lloc on comença l'execució del programa. Altrament, si es veu al panell esquerre, es mostren totes les funcions que executa el programa i és possible moure's per elles.
En aquest cas, l'autor, durant el procés de compilació, ha permès que es desin els símbols. Aquest fet, com es veurà a continuació, facilitarà molt la tasca. Podria haver no set així.
Quan IDA no troba símbols per la funció, la bateja amb el nom de sub_nombre. Probablement, totes aquestes funcions sense el nom es corresponguin a llibreries que incorpora el projecte. En aquesta pràctica no s'analitzaran totes, per tant, no és possible afirmar-ho amb certesa.

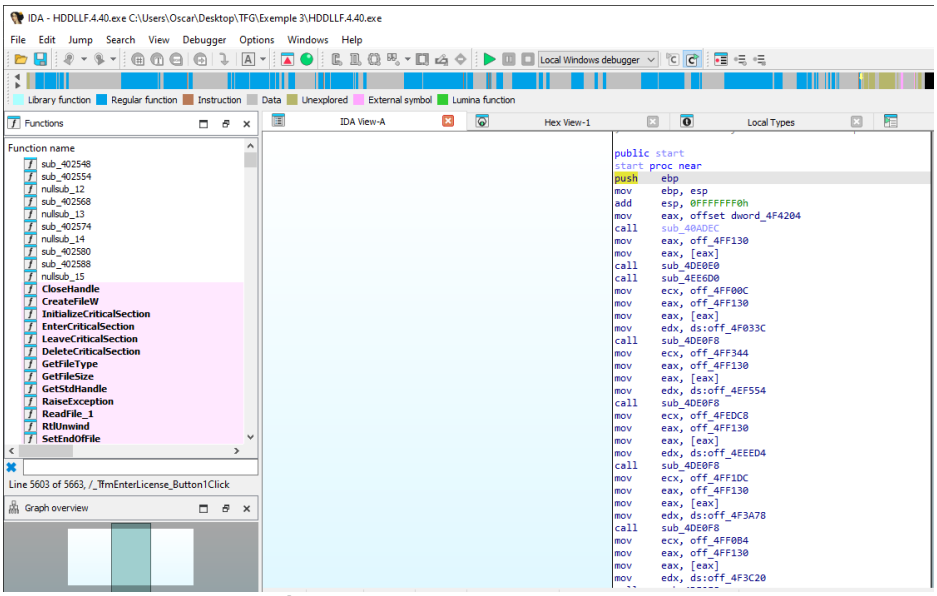

Figura 35: Cas pràctic 3: Primera vista amb IDA de HDDLLF4.40

Fent un parèntesis, fent menòria en els dos casos de laboratori anteriors, es van compilar desant els símbols (paràmetre -g de GCC). Si s'hagues carregat, el cas pràctic 1, per exemple, s'hagués mostrat el següent:

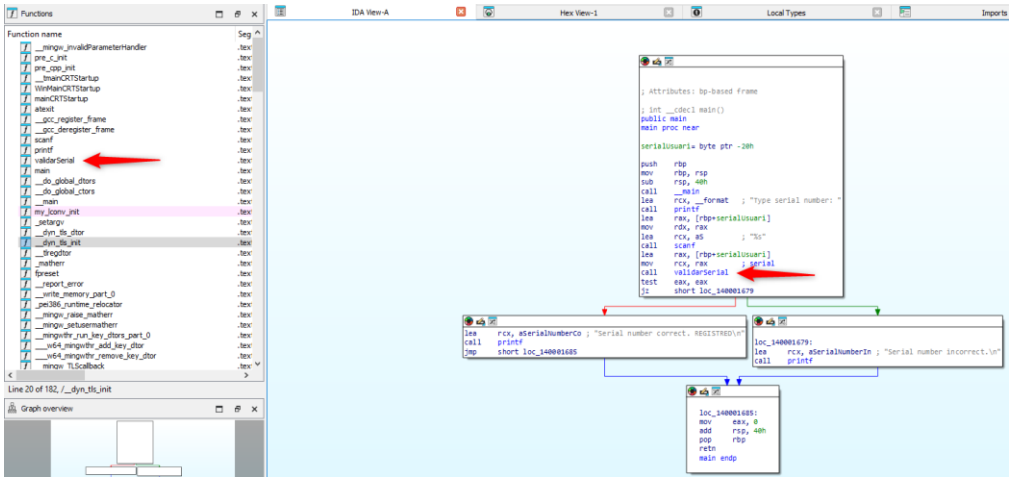

Figura 36: Carregant l'executable del cas pràctic 1 a IDA

Com es pot observar, entendre el funcionament hagués set immediat, Tanmateix, l'objectiu del laboratori era exposar alguns conceptes i no pas arribar a la solució el més ràpid possible.

Tornant al cas pràctic 3. Si s'aplica un filtre pel nom de la funció trobada anteriorment (TfmEnterLicense) es mostra el següent resultat:

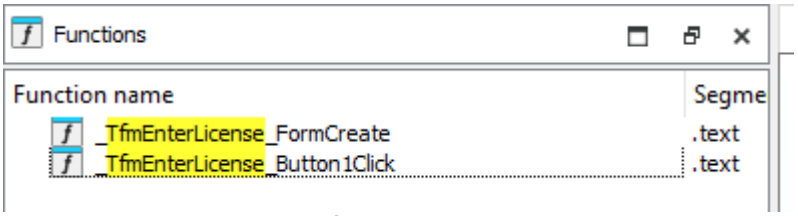

Figura 37: Cas pràctic 3: Filtrant les funcions

Per aquest cas interessa TfmEnterLicense_Button1Click, tal i com s'ha vist al editor de recursos. Al seu interior s'observa el següent:

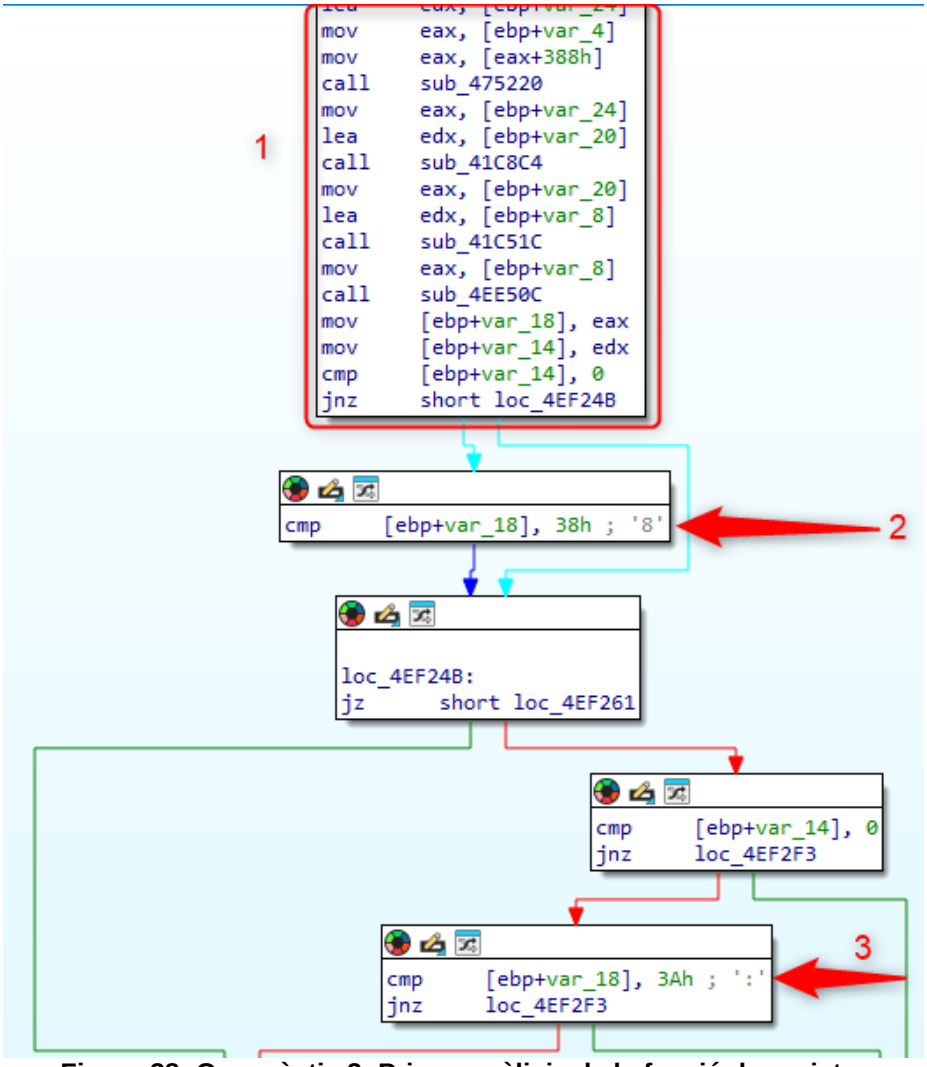

Figura 38: Cas pràctic 3: Primer anàlisis de la funció de registre

1. Es fan un seguit d'accions (més tard s'estudiaran). Al final de totes aquestes accions s'espera obtenir:
    - 38h: Registra.
    - 3Ah: Registra.
    - Qualsevol altre valor: No registra.
2. S'intueix que correspon a la modalitat "Home" o bé "Commercial" (més endavant s'estudiarà).
3. S'intueix que correspon a la modalitat que contraria a l'anterior.

Es poden comprovar aquestes afirmacions analitzant el codi que hi ha a la part inferior de la funció:

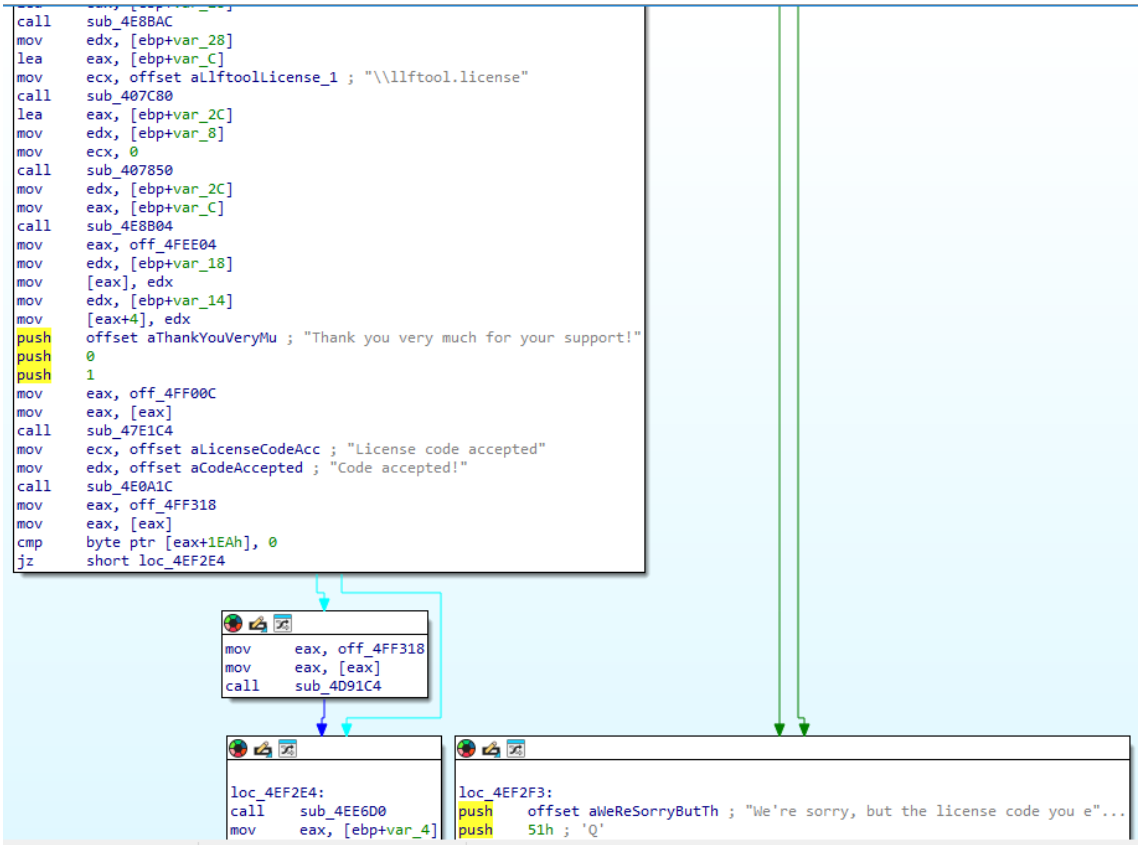

Figura 39: Cas pràctic 3: Analitzant que succeeix quan la clau és vàlida

D'aquí se'n pot extreure informació bastant valuosa sobre el que es produeix si la clau de registre valida:

- S'intueix que crea un arxiu llftool.license que molt probablement, conté la llicència. És interessant anotar aquest nom per un futur.
- Es creen diàlegs informant que el serial és vàlid i s'ha registrat l'aplicació. No obstant això, en cap moment es parla de la modalitat amb què s'ha registrat. El que fa sospitar que, un cop registrat, es reinicia l'aplicació i executa la funció de validació i, en funció del resultat es produeix l'elecció de la modalitat.

Per ara, se centraran els esforços a generar un número de sèrie vàlid. Tot i que es podria utilitzar IDA per explorar les funcions que es criden, s'utilitzarà x32dbg pel fet que es podrà analitzar els valors que es van generant de manera dinàmica. No sense abans obtenir una adreça de memòria propera a l'inici de la validació.

Per fer-ho es prem sobre una instrucció de l'inici de tot el procés de validació de la llicència per posicionar-s'hi i, a continuació, s'accedeix a la visualització de text:

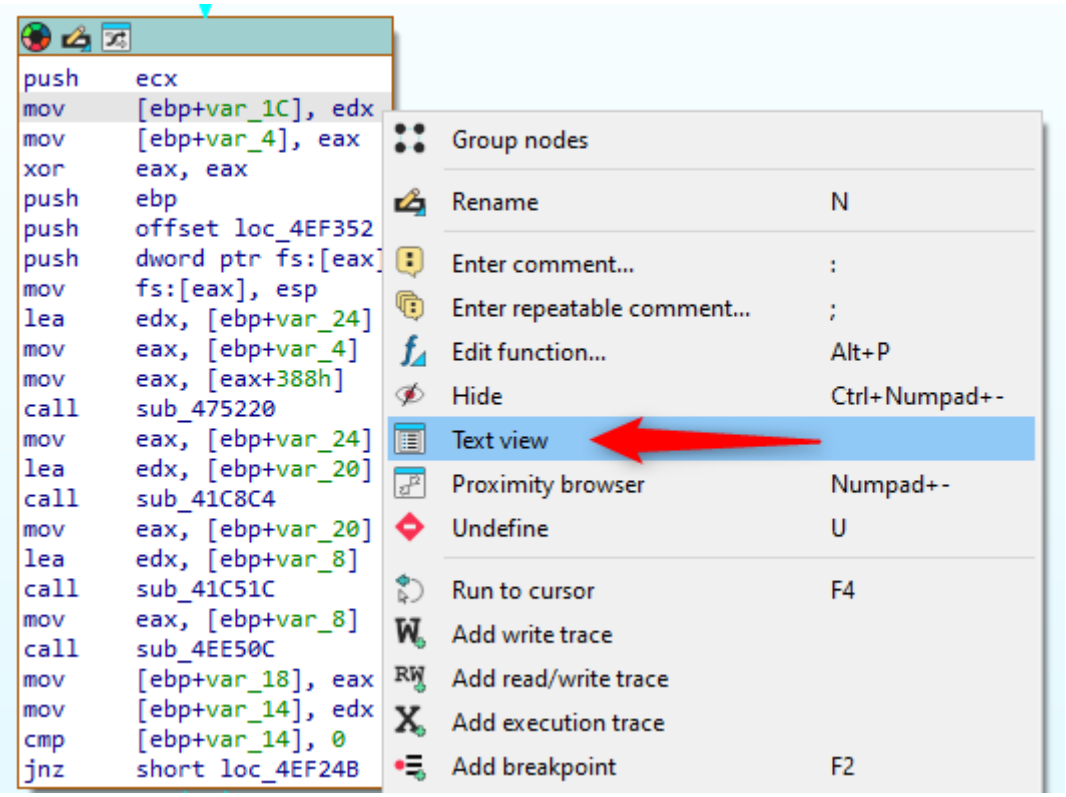

Figura 40: Cas pràctic 3: Alternant a la vista de text a IDA

Ara es pot obtenir l'adreça que s'utilitzarà a x32dbg per començar el procés:

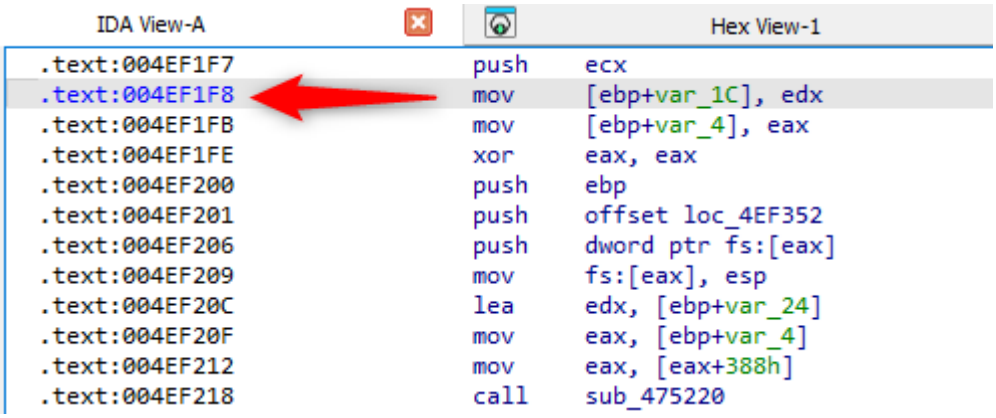

Figura 41: Cas pràctic 3: Obtenint l'adreça de l'inici de la validació

Seguidament, es procedeix a executar l'aplicació a x32dbg i s'estableix un punt d'interrupció a aquest adreça, s'introdueix una clau de registre (TFG-OALVAREZBA) i es prem el botó "Submit", de manera que s'aturarà la depuració just al principi de tot el procés.

Seguidament, per millor comprensió es canvia a moda gràfic. Allà es pot observar que, durant el procés de validació es criden quatre subrutines. Un cop finalitzades ja es dirimeix si la llicència és vàlida o no.

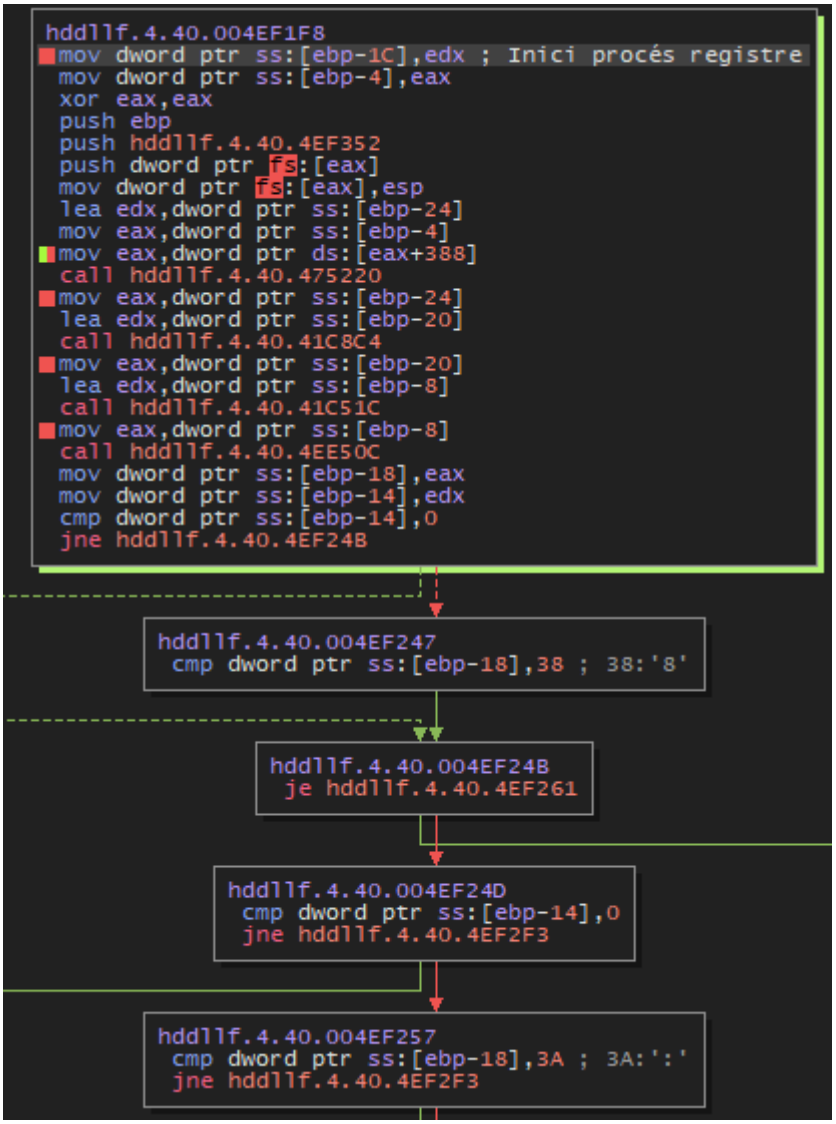

Figura 42: Cas pràctic 3: Funció de registre a x32dbg

Tal com s'ha anat veient, abans de cridar una funció, es preparen els registres pertinents amb els paràmetres, s'executa la funció i retorna el resultat al registre EAX (RAX a 64 bits).

Una bona estratègia és utilitzar l'opció "step over" per executar tota la funció sense analitzar-la pas a pas i veure si se'n pot extreure conclusions del seu propòsit. En cas contrari, sempre es pot reiniciar l'execució del programa i estudiar-la.

Si s'aplica aquesta tècnica a la primera crida, es pot observar que retorna el valor hexadecimal 'E' que, traduït a decimal, representa el nombre 14, que al seu torn, es correspon a la llargada de la cadena "TFG-OALVAREZBA". Aleshores ja es pot afirmar que aquesta funció conta la llargada de la cadena.

Figura 43: Cas pràctic 3: La primera crida conta la llargada de la cadena

Un cop s'ha examinat la primera funció, es pot procedir a analitzar la segona. Es pot veure que els paràmetres que se li passen són: la clau de registre introduïda a EAX i una adreça que apunta a la pila, amb tots els bits a 0 a EDX. Aquestes dades ofereixen poca informació substancial. A més, si s'aplica la mateixa tècnica que abans, és a dir, executar la funció i analitzar el seu retorn, es constata que retorna la clau de registre tal com s'ha introduït. D'altra banda, en examinar l'adreça de la pila, a la qual apuntava EDX, s'aprecia una còpia idèntica de la clau.
D'això només es pot deduir que EDX actua com a punter a una adreça de la pila, la qual actua com a destí després d'aplicar alguna operació desconeguda a la clau de registre. Aleshores, cal reiniciar l'execució i veure que succeïx:

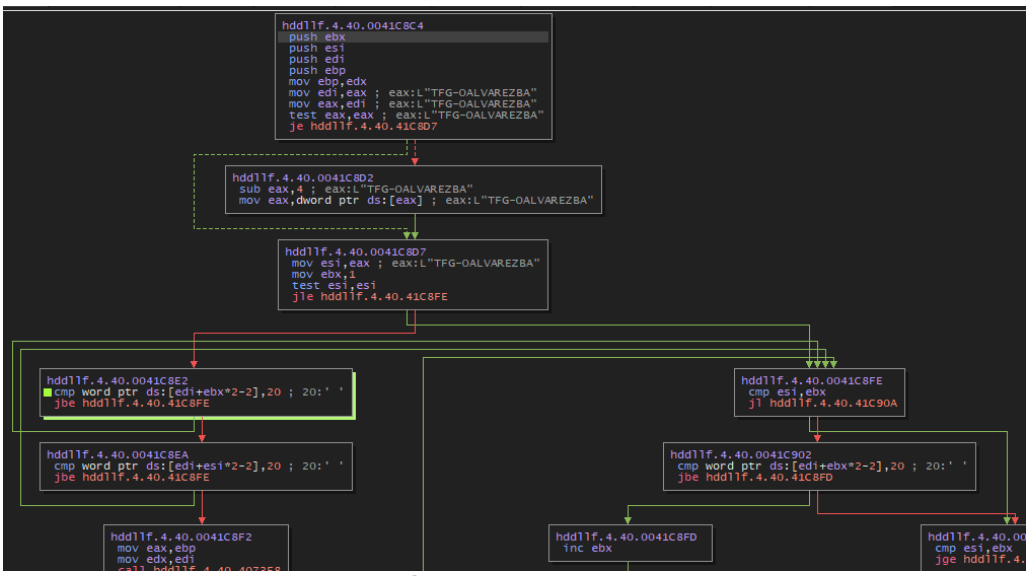

Figura 44: Cas pràctic 3: Analitzant la segona crida

De la captura anterior, es pot observar que el que s'està fent és comprovar si la primera o l'última posició de la clau entrada correspon amb el caràcter espai. En cas negatiu continua fins al final de la funció per retornar a la funció que l'ha cridat. En cas contrari, és a dir, que trobi un espai en blanc, li aplica un procés. Podem deduir, sense analitzar res més, que el que s'està fent és eliminar els espais en blanc tornar a calcular la mida de la cadena de text. Novament, es pot reiniciar l'execució de l'aplicació per comprovar si la teoria és certa.

Efectivament, si s'observa a la pila, a l'adreça de destí de la funció, es pot veure que, clarament ha eliminat els dos espais en blanc:

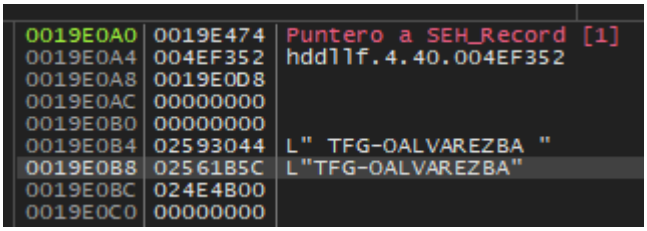

Figura 45: Cas pràctic 3: Demostració del funcionament de la segona crida

Bé, ara ja es coneix el propòsit d'aquesta funció. Tanmateix, abans d'avançar s'estudiarà com funciona les instruccions `cmp word ptr ds:[edi+ebx*2-2], 20` i `cmp word ptr ds:[edi+esi*2-2], 20` utilitzades per posicionar-se sobre el primer i l'últim caràcter respectivament, ja que aquest mecanisme resulta habitual en tractament de cadenes de text:

En primer lloc, un cop situats a la primera de les instruccions ja mencionades, cal observar com s'han preparat els registres en les instruccions anteriors:

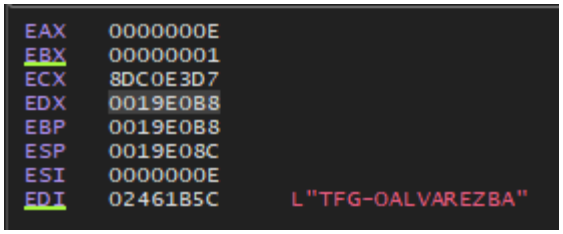

Figura 46: Cas pràctic 3: Estat dels registres en el posicionament de les cadenes de text

Aleshores, calculant l'adreça `cmp word ptr ds:[edi+ebx*2-2], 20` s'obté que ebx * 2 – 2 = 0 i, per tan, la instrucció es tradueix a `cmp word ptr ds:[02461B5C], 20`. Analitzant la instrucció:

- cmp: és una instrucció de comparació de x86.
- word ptr: significa la mida de l'operand, en aquest cas 16 bits (2 bytes).
- ds: indica on es troba la informació. En aquest cas segment de dades (data segment). És a dir a la memòria.
- [@] adreça de la memòria a la qual es fa referència.

Efectivament, analitzant la memòria, correspon al primer caràcter (lletra T). Al comparar-la amb un espai, no és igual, i passa a analitzar l'última posició.

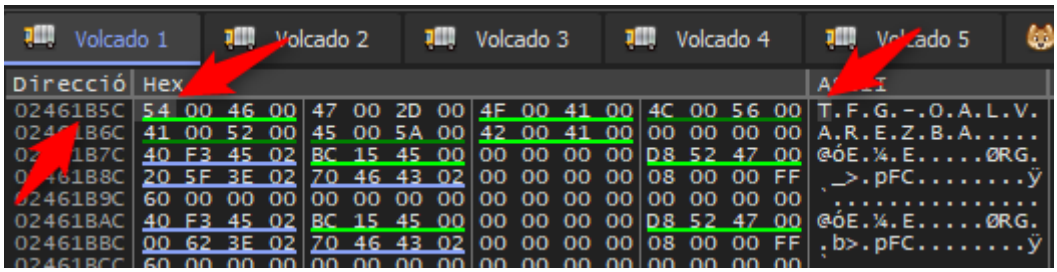

Figura 47: Cas pràctic 3: Adreça calculada de la primera posició de la cadena a la memòria.

De manera anàloga es calcula l'adreça obtenint: `cmp word ptr ds:[2461B76], 20`:

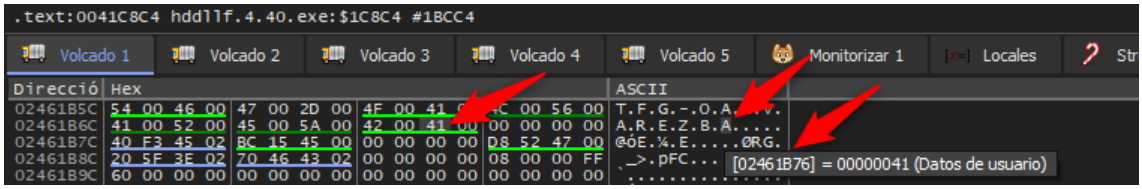

Figura 48: Cas pràctic 3: Adreça calculada de l'última posició de la cadena a la memòria.

És en aquest cas és interessant veure que:

- L'adreça de EDI indica l'inici de la cadena de text a la memòria.
- ESI conté la mida de la cadena la qual es multiplica per 2. Si es veu com està desada la cadena, s'aprecia que es reserven 2 bytes per cada caràcter.

Si se suma ESI a EDI, s'apunta al final de la cadena. Malgrat això, interessa apuntar a un caràcter abans de finalitzar la clau, per veure si és un espai en blanc o no. Aleshores, es resten dos bytes per aconseguir-ho.

Vist això es pot procedir a analitzar la tercera i penúltima de les funcions. És un cas similar a l'anterior: Se li passa la clau al registre EAX i també una adreça buida de la pila a on aparentment s'utilitza de contenidor. Si es fa servir l'estratègia d'executar-la per poder fer deduccions d'acord amb el que retorna, tampoc s'obté cap resultat concloent (la mateixa clau de llicència al registre EAX i a l'adreça destinada a ser el contenidor a la pila). A les hores, novament, cal explorar-ne el propòsit:

Figura 49: Cas pràctic 3: Analitzant la tercera crida

En aquest cas, el funcionament no és tan evident com en l'anterior funció. A simple vista s'aprecia l'estructura d'un bucle el qual fa pensar que es fa alguna acció per cada caràcter de la cadena. Ni ha prou d'executar pas per pas un parell de cicles per adonar-se que és d'aquesta manera. Si s'exhaureixen tots els caràcters, es fa avinent que en cap cas s'executa la instrucció xor ax, 20. Així com tampoc es fa cap modificació a la clau original i, per aquest motiu, s'obté la clau original de retorn.

Per entendre el propòsit d'aquesta funció, cal fixar-se en el següent bloc el qual determina la condició perquè s'executi aquesta operació xor:

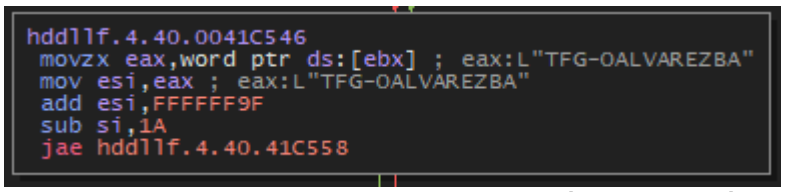

Figura 50: Detall del codi per convertir de minúscula a majúscula

Sigui T el primer caràcter de la clau de registre introduïda:

- `movzx` (move zero extended), mou el primer caràcter “T” al registre EAX i exten amb zeros el que li falta per arrivar a 32 bits.
- Mou el contingut de EAX (el caràcter “T”) al registre ESI.
- Suma FFFFFF9F a ESI que conté el valor 54 (lletra “T”) = FFFFFFF3
- Resta 1A als últims 16 bits del registre ESI.
- `jae` comprova si està actiu el CF (carry flag).

Com que no es produeix sobreeiximent (carry), no s'executa el bloc de l'XOR. Ara el lector pot pensar, per quins caràcters es produeix? Si es revisa la taula ASCII, es pot veure que si se suma el caràcter "a" el qual equival a 61 en hexadecimal es produeix el següent:

- FFFFFF9F + 61 = 1 0000 0000. Ara si hi ha desbordament i, com que el registre és de 32 bits passa a valer 00000000. Aleshores s'activa ZF (zero flag) i CF (carry flag).
- Es resta 1A als últims 16 bits = FFFF FFE6. Ara el ZF torna a estar a 0.
- `jae` avalua CF. Com que CF està actiu, ara sí que s'executa la XOR. De fet, ho farà per tots els caràcters ASCII majors a 61 hexadecimal.

Un altre fet interessant és entendre que fa aquesta XOR. Per veure-ho s'analitzarà un cas: caràcter "a" que ja hem vist que equival a 61 hexadecimal.

```
61h 0110 0001
XOR
20h 0010 0000
-------------
41h 0100 0001
```

S’ha obtingut el valor 41 hexadecimal, que si s’observa, novament, a la taula ASCII, es correspon amb el caràcter “A”. Aleshores, ja s’entén el propòsit d’aquesta funció: convertir els caràcters de minúscula a majúscula.

Ara el lector es podria qüestionar, i perquè una XOR en lloc d'una simple resta? La resposta és perquè XOR és més òptim internament que la instrucció `sub`. A grans trets, sense aprofundir molt en la qüestió, en fer una XOR, internament, només cal comparar bits per aplicar-la. Mentre que una resta implica fer sumes i, depenent si es treballa amb signes, complements a dos.

A tall d'observació, el caràcter "-", utilitzat habitualment en les claus de registre, així com els números tenen una representació en hexadecimal menor a 61. Aleshores no els hi fa cap transformació.

Després d'analitzar la tercera funció, ja només queda la quarta i última. Fins ara, s'ha vist com l'aplicació fa un sanejament de les dades: llegeix la clau i en calcula la mida, n'elimina els espais en blanc d'inici i fi i converteix la clau de la llicència en majúscula. Ara ja només resta esperar la funció on es produeix la validació en si mateixa.

Si s'executa pas a pas, es veu com es compara la mida de la clau amb el valor 13 hexadecimal per fitar-ne la mida exacta. La clau usada fins ara, TFG-OALVAREZBA, està formada per 14 caràcters, tal com s'ha vist anteriorment, una llargada de "E" en hexadecimal. Al ser menor, pren el salt fins al final i s'acaba la funció.

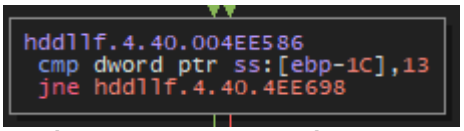

Figura 51: Cas pràctic 3: Comprovació de la llargada de la clau

En aquest punt s'obté el primer dels requisits i obliga a modificar la clau. A partir d'ara s'utilitzarà "TFG-OALVAREZBA23456". Cal veure que s'afegeixen caràcters no repetits. Es decideix així per tal de facilitar la detecció de qualsevol acció en el tractament dels caràcters de la clau.

Seguidament, s'observa en el codi que es fa referència a un punt de memòria on hi ha una cadena alfanumèrica bastant sospitosa. És una bona idea anotar-la abans de continuar.
A banda d'això, si s'aprofundeix una mica més en l'anàlisi visual del codi, s'observa que hi ha quatre blocs de codi on es fa la mateixa acció: Se selecciona un caràcter i després es criden dues funcions les quals, per ara, se'n desconeix el propòsit, no obstant això, sí que és important veure que només es fa aquest procés en quatre caràcters de la clau i, per tant, tot apunta al fet que la resta no intervenen en el procés de validació.

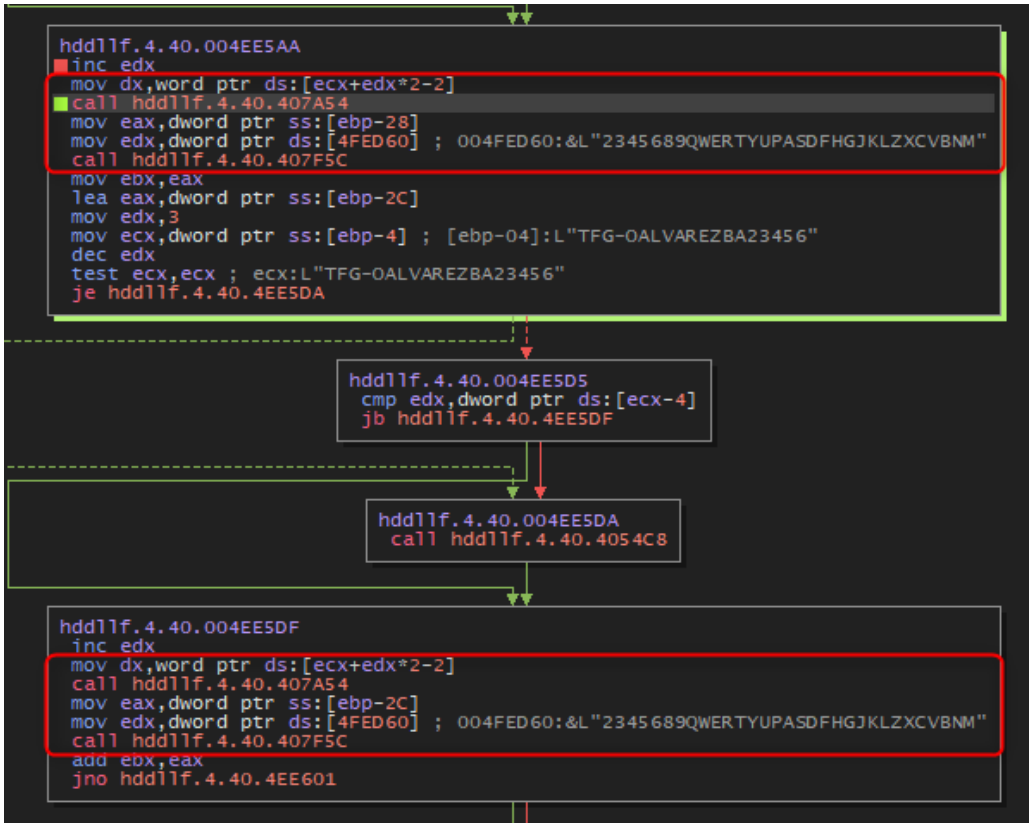

Figura 52: Cas pràctic 3: Mateixa acció amb determinats caràcters de la clau.

Dit això, es procedeix a analitzar que succeeix:

En primer lloc, abans d'executar una crida, com ja s'ha anat veient, es preparen els paràmetres. En aquest cas és important veure que el paràmetre és la lletra "F"; el segon caràcter de la clau.

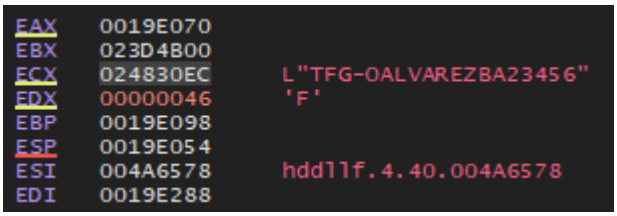

Figura 53: Cas pràctic 3: Paràmetres abans d'executar la funció 407A54.

Si s'executa la primera crida (407A54) per sobre, s'aprecia que retorna una adreça de memòria la qual conté el caràcter "F". És a dir, extreu el caràcter de tota la cadena per ser utilitzat per la següent crida.

En la següent de les crides mencionades, si s'executa per sobre i es mira que ha retornat, amb una mica d'atenció, és possible adonar-se que el seu propòsit és comptar la posició que representa el segon caràcter de la clau a la cadena de text que s'ha anotat anteriorment: 13 hexadecimal equival a 19 decimal, que efectivament coincideix amb la posició de "T" a la següent cadena: 2345689QWERTYUPASDFHGJKLZXCVBNM

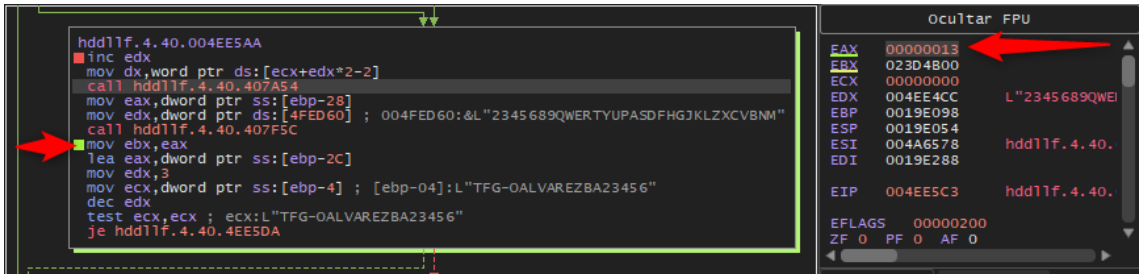

Figura 54: Cas pràctic 3: La funció conta la posició del caràcter dins la cadena de text

Seguidament, es mou el resultat al registre EBX i es procedeix a repetir exactament el mateix procés amb el tercer caràcter de la clau. Cal fixar-se que aquest cop en lloc de moure el resultat a EBX, li suma.

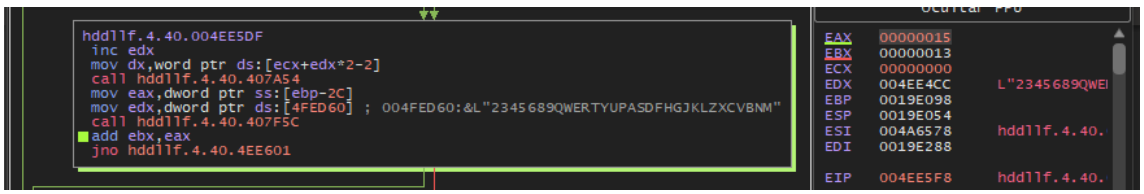

Figura 55: Cas pràctic 3: Repetició del procés amb tercer caràcter

Observant els altres dos blocs similars, es descobreix que es fa el mateix amb el penúltim i antepenúltim caràcter de la clau.

Ara ja es coneix el mecanisme de validació: utilitza els caràcters de les posicions 2, 3, 17 i 18 de la clau de registre i els busca dins la cadena de text, que ja incorpora el mateix codi com a constant, sumant aquests valors. Addicionalment, s'ha de tenir en compte que al final del procés, es resta quatre unitats a aquesta suma.

Figura 56: Cas pràctic 3: Suma de quatre unitats al final del procés

Ara, el lector ja està en disposició de generar clau de registre vàlida. Tanmateix, abans, es veuran unes consideracions:

- La cadena no incorpora tots els nombres de l'1 al 9, hi falta l'1 i el 7 ni tampoc hi ha totes les lletres de l'abecedari, ja que hi falta la "I" i la "O".
- Si s'usa algun d'aquests caràcters la funció que conta la posició del caràcter a la cadena de text retorna zero.

Tot i que es podria provar de jugar amb aquest fet per generar claus vàlides. Només es faran servir caràcters "permesos".

A continuació es genera una clau vàlida per la primera de les modalitats el valor 38 hexadecimal:

- 38 hexadecimal = 56 decimal.
- Si se li sumen 4 unitats que es resten al final a 56, s'obté el valor 60 decimal.
- Si es dividiex per 4 caràcters, cal seleccionar cuatre vegades el caràcter en la posició 15 (la lletra "P").

I, efectivament, la clau TPP-OALVAREZBA23PP6, valida:

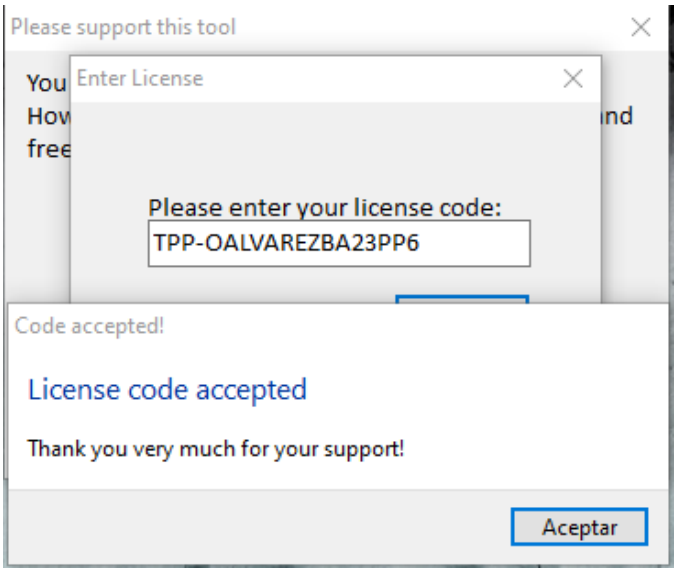

Figura 57: Cas pràctic 3: Registrant l'aplicació

I, es pot comprovar que ho fa amb la modalitat “Home”

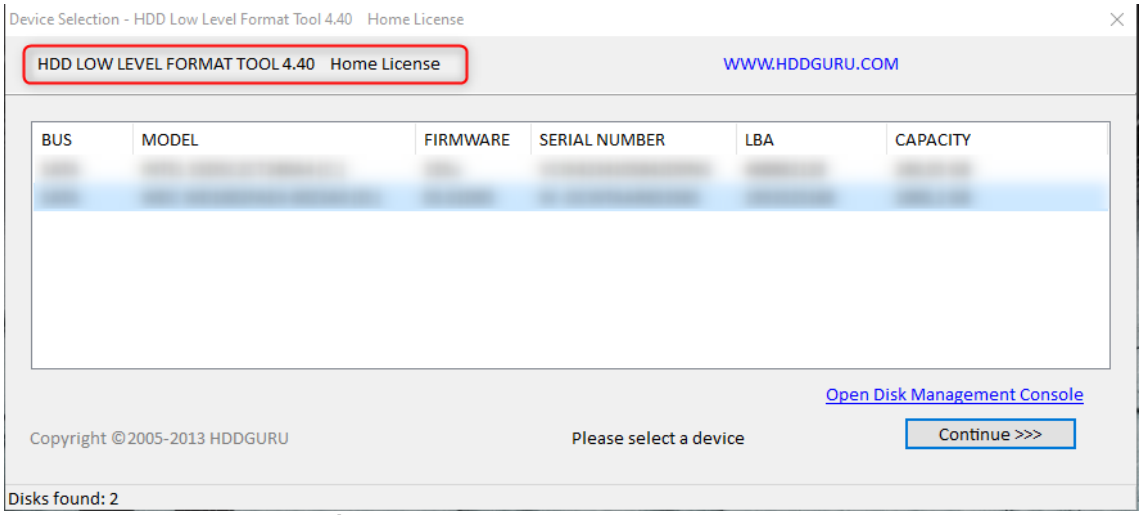

Figura 58: Cas pràctic 3: 58 hexadecimal correspon a la modalitat Home

Aleshores, per concloure, cal generar una clau que validi per 3A hexadecimal.

- 3A hexadecimal = 58 decimal.
- Si se li sumen 4 unitats que es resten al final, s’obté el valor 62 decimal.
- Si s’observa que són dues unitats més que el cas anterior, es pot imposar la següent clau de registre: TPP-OALVAREZBA23PS6.

Ara, és un bon moment per recordar a on es va desar l’arxiu de l’acceptació dels termes i veure si, efectivament s’ha desat la clau allà:

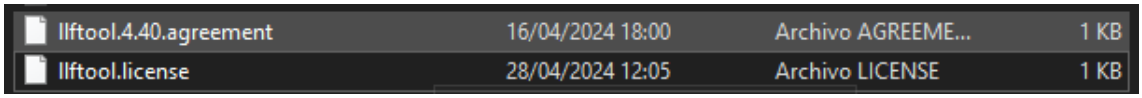

Figura 59: Cas pràctic 3: Ubicació de la clau al sistema

Si s’edita l’arxiu llftool.license amb un editor de text i s’introdueix la nova clau anteriorment trobada, efectivament, registra la modalitat “Commercial”.

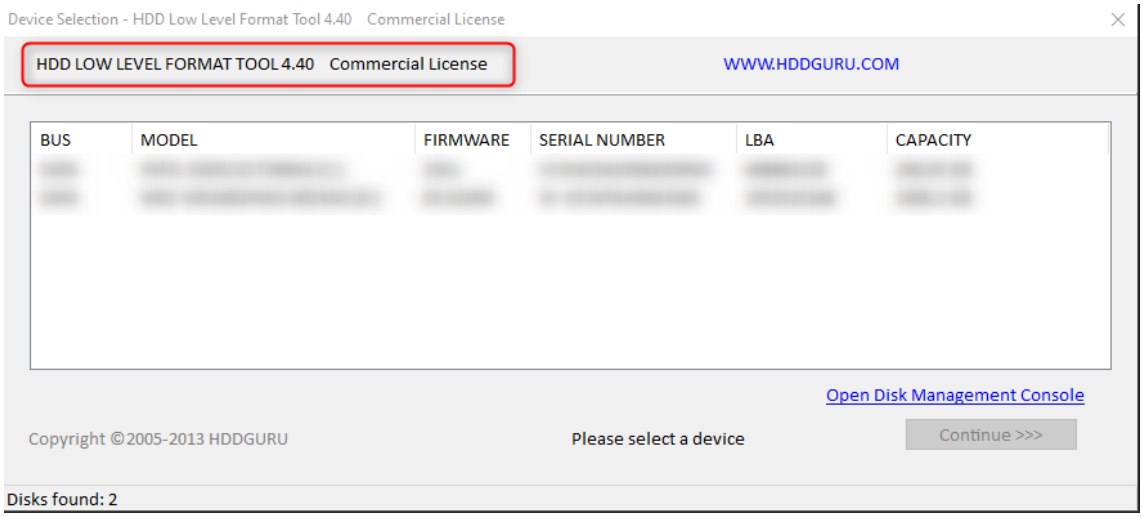

Figura 60: Cas pràctic 3: 3A hexadecimal correspon a la modalitat Commercial

Sabent com funciona el mecanisme de registre, fins i tot es pot aconseguir un aspecte més similar a les claus de registre "habituals":

- TPPO-ALVA-REZB-APP4 per la modalitat “Home”.
- TPPO-ALVA-REZB-APS4 per la modalitat “Commercial”.

A tall informatiu, si es torna al punt d'entrada (Entry Point) i es mira les crides que es fan en iniciar l'aplicació, més concretament la crida sub_4EE6D0, s'aprecia com llegeix l'arxiu llftool.license i executa el mecanisme ja conegut per triar la modalitat i executar l'aplicació amb limitacions o sense. Per tal de no estendre més aquest document es deixa en mans del lector comprovar-ho.

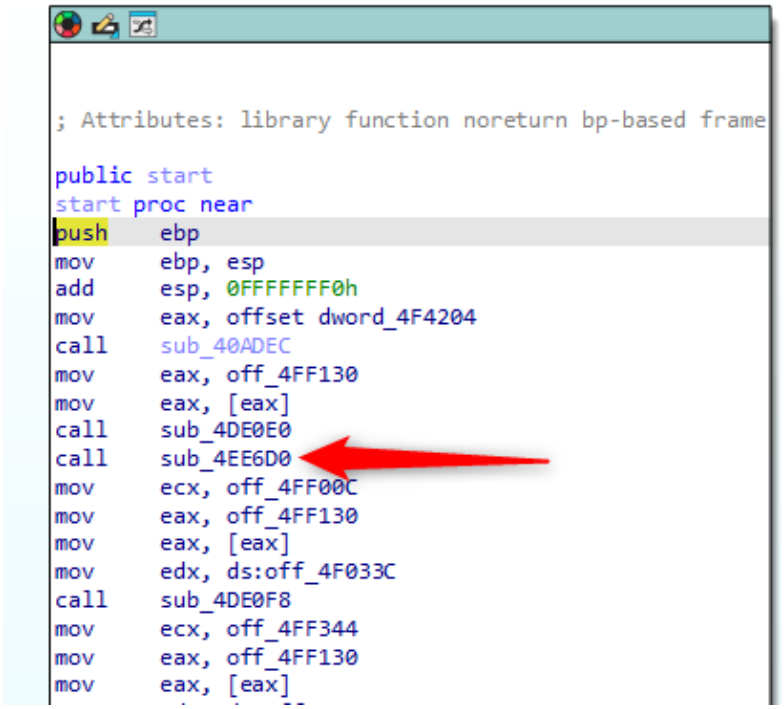

Figura 61: Cas pràctic 3: Funció que comprova la llicència la iniciar

Finalment, igual que en l’exemple anterior es desenvolupa el generador de claus. En aquest cas també es farà amb llenguatge C:

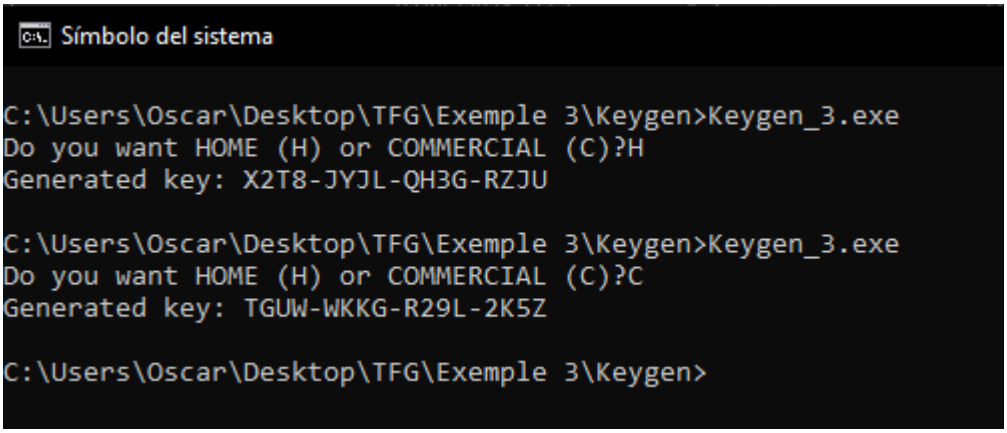

Figura 62: Cas pràctic 3: Generador de claus

Addicionalment, i com a últim pas, cal eliminar l'arxiu llftool.licesne, ja que, de moment, no s'ha adquirit cap clau legalment i, en cas d'haver-la adquirit, s'utilitzarà la que proporcioni el desenvolupador.

3.4. Cas pràctic 4:

En aquest cas, tal com s'ha comentat es parteix del codi descompilat. Si s'exploren els arxius obtinguts, es pot veure que es generen tres carpetes:

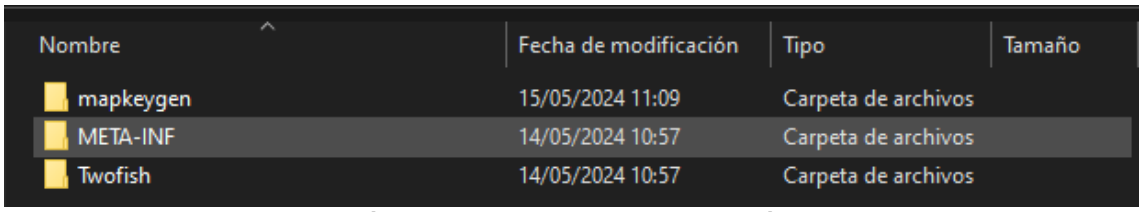

Figura 63: Cas pràctic 4: Arxius obtinguts després de descompilar

1. mapkeygen: Conté una classe amb el codi amb l'algorisme per fer la validació del codi d'activació i un altre que conté el codi necessari per a la interfície gràfica.
2. META-INF: Conté l'arxiu MANIFEST el qual dona informació de l'estructura de les classes.
3. Twofish: Conté una classe amb el codi de l'algorisme de xifratge Twofish.

El primer que cal fer després de descompilar el codi és donar una ullada superficial al codi descompilat i veure que no conté cap codi que faci alguna acció que pugui infectar el nostre equip. Cal recordar que no es té cap mena de certesa ni garantia que aquests activadors i generadors de claus descarregats de fonts desconegudes, siguin estrictament el que prometen. Ja s'avança que no és el cas i, el primer que es farà és executar-lo:

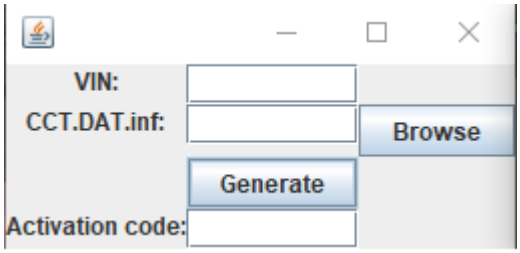

Figura 64: Cas pràctic 4: Generador de claus executat

En aquest punt el lector es pot preguntar: En aquest cas s'ha pogut descompilar i accedir al codi, que succeeix si no és possible? Existeixen recursos a Internet com VirusTotal[24] on es pot carregar l'arxiu en qüestió i s'analitza amb una gran quantitat d'antivirus.

[24]https://www.virustotal.com/gui/home/upload (Data de consulta 27/05/2024)

En la captura anterior es veu que cal el VIN (Vehicle Identification Number) o el que és conegut comunament com en número de bastidor. Aquest número apareix escrit, entre altres, en algun lloc del xassís cotxe. I una ruta a un arxiu que conté el propi mapa.

- Per aquest exemple, com que l'objectiu és purament el científic, copiaré l'arxiu necessari i en descartaré la resta. De la mateixa pàgina web[25] on s'ha obtingut l'activador també hi ha els mapes que caldria copiar i posteriorment activar des del propi navegador del vehicle.
- Respecte al VIN, en generaré un d'inventat per aquest exemple: VF3CRBHY6JT012345.

Si s'executa l'activador, efectivament s'obté la cadena que cal utilitzar per activar les cartografies:

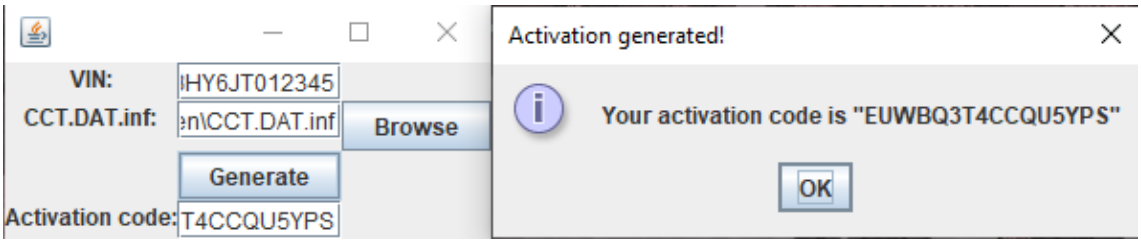

Figura 65: Cas pràctic 4: Codi d'activació generat

Vist això, es procedeix a analitzar el codi prèviament descompilat per entendre'n el funcionament. A tall informatiu, es pot veure que, al descompilar, totes les variables han pres un nom genèric. Abans de començar s'han anomenat amb un nom més descriptiu per fer-ne l'estudi més fàcil.

Al prémer el botó "Generate" s'executa la funció "generate" on se li passa per paràmetre la ruta de l'arxiu (CCT.DAT.inf) i el VIN.

```
24      public static String generate(File file, String vin) throws IOException, InvalidKeyException {
25          byte[] mapCode = new byte[9];
26          byte[] vinCode = new byte[12];
27          byte[] activationKey = new byte[17];
28          byte[] combinedData = new byte[33];
29          byte[] encryptionKey = new byte[16];
30          int checksum = 0;
31
32          if (vin.length() < 17) {
33              throw new RuntimeException("VIN too short");
34          } else {
35              FileInputStream fis = new FileInputStream(file);
36
37              try {
38                  fis.read(mapCode, 0, 8);
39              } finally {
40                  fis.close();
41              }
42
43              mapCode[8] = 0;
44              // mapCode ara conté {98, 101, 97, 100, 101, 102, 101, 48, 0} ('beadefe0')
```

Figura 66: Cas pràctic 4: Codi del generador de claus part 1

[25]https://antoniocarrascoblog.wordpress.com/2022/07/01/instalar-mapas-2022-1-v118-0-completo-europa-smeg-smeg-smeg-iv2-rt6-emyway-o-wipnav/ (Data de consulta 27/05/2024)

El primer que fa és comprovar que el VIN té la llargada que s'espera. Si aquest fet es compleix, llegeix els 8 primers caràcters de l'arxiu passat per paràmetre i li posa un zero al final per indicar final de cadena.

Cal observar que desa els caràcters llegits en dins d'un vector de "bytes" i no dins d'un vector de "Strings". Aleshores, el que es desa a cada posició del vector és la conversió a decimal de la taula ASCII del caràcter en qüestió.

Seguidament, es procedeix a la lectura dels últims 9 dígits del VIN i es van convertint de “String” a “byte”. Cal veure que també va fent una suma d'aquests valors i els desa a la variable checksum per, un cop completada aquesta suma, fer-li el complement a 2 i, al resultat, aplicar-li una AND lògica amb el valor 255. Del valor obtingut en aquesta última operació se n'aprofiten els dos primers valors els quals s'incorporen a les dues últimes posicions de vinCode prèvia conversió a tipus byte.

Com a curiositat, cal veure que l'autor va imprimir les variables mapCode i vinCode per pantalla, segurament a tall de depuració del codi i se'ls hi va descuidar. No tenen més interès en el funcionament del programa.

```
45
46      for (int i = 0; i < 9; ++i) {
47          byte vinByte = vin.getBytes()[i + 8];
48          vinCode[i] = vinByte;
49          checksum += vinByte;
50      }
51      // vinCode ara conté {54, 74, 84, 48, 49, 50, 51, 52, 53, 0, 0, 0}
52      // checksum ara conté 515 (54 + 74 + 84 + 48 + 49 + 50 + 51 + 52 + 53)
53
54      checksum = ~checksum + 1 & 255;
55      // checksum ara conté 253
56
57      char[] checksumChars = String.format("%02d", checksum).toCharArray();
58      // checksumChars ara conté {'2', '5', '3'}
59
60      vinCode[10] = (byte) checksumChars[0];
61      vinCode[11] = (byte) checksumChars[1];
62      // vinCode ara conté {54, 74, 84, 48, 49, 50, 51, 52, 53, 0, 50, 53}
63
64      System.out.println("map_code=" + new String(mapCode));
65      System.out.println("vin_code=" + new String(vinCode));
```

Figura 67: Cas pràctic 4: Codi del generador de claus part 2

Després de les anteriors modificacions al VIN, copia les primeres nou posicions de la variable vinCode (cal veure que s'ignoren les tres últimes posicions) a les primeres posicions de combinedData per duplicat, tal com es pot veure a la següent captura. Tot seguit fa el mateix amb les set primeres posicions del codi del mapa (obtingut de CCT.DAT.inf) i acaba aplicant un parell més de transformacions dels valors. Finalment, combinedData conté els valors {54, 54, 74, 74, 84, 84, 48, 48, 49, 49, 50, 50, 51, 51, 52, 52, 53, 53, 98, 98, 101, 101, 97, 97, 100, 100, 101, 101, 102, 102, 101, 48, 0}.

```
67    for (int i = 0; i < 9; ++i) {
68       combinedData[2 * i] = vinCode[i];
69       combinedData[2 * i + 1] = vinCode[i];
70    }
71    // combinedData ara conté {54, 54, 74, 74, 84, 84, 48, 48, 49, 49, 50, 50, 51, 51, 52, 52, 53, 53,
72
73    for (int i = 0; i < 7; ++i) {
74       combinedData[18 + 2 * i] = mapCode[i];
75       combinedData[18 + 2 * i + 1] = mapCode[i];
76    }
77    // combinedData ara conté {54, 54, 74, 74, 84, 84, 48, 48, 49, 49, 50, 50, 51, 51, 52, 52, 53, 53,
78
79    combinedData[31] = mapCode[7];
80    combinedData[32] = 0;
81    // combinedData ara conté {54, 54, 74, 74, 84, 84, 48, 48, 49, 49, 50, 50, 51, 51, 52, 52, 53, 53,
```

Figura 68: Cas pràctic 4: Codi del generador de claus part 3

En el següent pas es genera un clau pel xifrat i es crida la funció encrypt. Aquesta funció, xifra amb l'algoritme Twofish les dades preparades en el pas anterior (combinedData) i, al resultat obtingut, li aplica una XOR element a element amb la mateixa clau de xifrat:

```
83    for (int i = 0; i < 16; ++i) {
84       encryptionKey[i] = INIT_VECTOR[i];
85    }
86    // encryptionKey ara conté {-83, 44, -122, -99, -8, -54, -85, 85, 17, 42, 30, -126, -103, 108, 118
87
88    byte[] encryptedData = encrypt(combinedData, encryptionKey);
89    // encryptedData ara conté {57, -94, -45, 116, -56, -11, -100, 11, -6, -7, 3, -18, -110, 101, 33,
```

Figura 69: Cas pràctic 4: Codi del generador de claus part 4

A tall informatiu, per aconseguir el valor que retorna la funció “encrypt”, he creat un projecte JAVA amb el NetBeans i he modificat el codi per tal que no vagi a buscar el codi del mapa a l’arxiu CCT.DAT.inf, sinó que li passo directament per paràmetre des del "main" en un "String" juntament amb el VIN i, posteriorment, el converteixo a un vector de tipus byte. D'aquesta manera es pot imprimir el resultat per la sortida de la consola:

```
    public static void main(String[] args) throws IOException, InvalidKeyException {
        //new MapKeygenGui();
        generate("beadefe0","VF3CRBHY6JT012345");
    }
}
```

Figura 70: Cas pràctic 4: Funció main modificada a NetBeans

A continuació s’aplica un altre xifrat. S’utilitza les dades xifrades anteriorment per generar una nova clau de xifrat. I, al seu torn, es modifica un altre cop la variable “combinedData” copiant la meitat superior del vector a la meitat inferior. Tal i com s’ha comentat s’utilitzen aquest nou parell de dades per aplicar el procés de xifrat i la posterior XOR sobre les dades xifrades.

```
90
91    for (int i = 0; i < 16; ++i) {
92       encryptionKey[i] = encryptedData[i];
93       combinedData[i] = combinedData[i + 16];
94    }
95    // encryptionKey {57, -94, -45, 116, -56, -11, -100, 11, -6, -7, 3, -18, -110, 101, 33, -44}
96    // combinedData ara conté {53, 53, 98, 98, 101, 101, 97, 97, 100, 100, 101, 101, 102, 102, 101, 48
97
98    encryptedData = encrypt(combinedData, encryptionKey);
99    // encryptedData ara conté {96, 124, 61, -37, 117, 74, 123, -51, -103, -35, -11, -71, 10, -64, -82
```

Figura 71: Cas pràctic 4: Codi del generador de claus part 5

Finalment, agafa cada posició del vector amb les dades xifrades de forma definitiva, hi aplica un desplaçament d'un bit cap a la dreta i, seguidament, al resultat, s'hi aplica una AND amb el valor 63. El resultat d'aquesta operació coincideix amb un caràcter de la cadena MAP, el qual es desa com un element de tipus byte a la variable "activationKey".

Per acabar, es mostra "activationKey" com un String que correspon a la clau de xifrat.

```
101 ▼  for (int i = 0; i < 16; ++i) {
102        activationKey[i] = (byte) MAP.charAt(encryptedData[i] >> 1 & 63);
103    }
104
105    activationKey[16] = 0;
106    // activationKey ara conté {69, 85, 87, 66, 81, 51, 84, 52, 67, 67, 81, 85, 53, 89, 80, 83, 0}
107
108    System.out.println("Activation key: " + new String(activationKey));
109    return new String(activationKey); //EUWBQ3T4CCQU5YPS
```

Figura 72: Cas pràctic 4: Codi del generador de claus part 6

Amb la finalitat de clarificar l'últim procés es mostrarà l'anàlisi detallada pel primer caràcter de la clau de registre:

1. 96 decimal = 01100000 binari.
2. S'aplica el desplaçament d'un bit a l'esquerra: 00110000.
3. 63 en base decimal es tradueix com: 00111111 en binari.
4. En aplicar 00110000 AND 00111111 = 00110000 = 48 en base decimal.
5. Si se cerca la posició 48 a MAP, correspon al caràcter "E" (Tenint en compte que les posicions es conten de 0 a 63).

Si es repeteix aquest procés per cadascun dels valors de "encryptedData", efectivament la clau d'activació trobada: EUWBQ3T4CCQU5YPS. Queda a voluntat del lector fer la comprovació cas per cas.

4. Conclusions i treballs futurs

Per concloure, tal com s'ha vist anteriorment, una aplicació que valida la clau en el mateix equip, sense cap intervenció d'un servidor remot, queda completament vulnerable. N'hi ha prou amb estudiar el seu codi mitjançant un desassemblador i/o un debugger per obtenir el sistema de validació i poder crear una clau vàlida o alterar el codi per saltar la protecció.

Sí que és cert que hi ha maneres per ofuscar el codi i proteccions antidebug, però només són dificultats afegides. L'aplicació continua sent vulnerable de la mateixa manera. A continuació, tot i que no es pretén crear una llista exhaustiva, es detallen algunes mesures de protecció utilitzades i que es proposen per estudiar en treballs futurs:

Ofuscació de codi: Consisteix a transformar el codi de manera que sigui més complicat de llegir o analitzar. Per exemple, si considerem un codi en un llenguatge de programació, com ara C, C++ o JAVA, una manera d'aplicar aquesta tècnica seria canviar els noms de les variables per noms sense sentit, afegir codi inútil que no es crida mai, codi que fa transformacions a les dades i les retorna al seu estat original al seu estat original i/o funciones funcions afegides que l'únic que fan és acabar cridant la funció que ja es pretenia cridar. Les tècniques citades anteriorment només en són alguns exemples. Sí que és veritat que no impedeix l'enginyeria inversa, però, tal com es pot veure, n'entorpeix bastant la seva anàlisi. Especialment si ho fem en llenguatge assemblador.

Marques d'aigua al codi: Consisteix a generar "hashes" de parts del codi per verificar si s'ha modificat amb la intenció de saltar el mecanisme de registre i, si existeixen aquestes modificacions, prendre les mesures que el desenvolupador cregui oportunes. Per exemple, no deixar executar l'aplicació. Aquests "hashes" poden ser presents en el mateix programa o ser accedits a un servidor remot en el moment de la seva execució. Tampoc n'impedeixen l'estudi ni la modificació, però sí que la dificulten, ja que també cal modificar la part del codi que realitza aquesta comprovació per tal d'evadir-la.

Validació d'integritat dels arxius: En la línia del cas anterior, se signen o es genera un "hash" dels arxius que es volen protegir i, en cas de detectar una modificació el desenvolupador pot preparar quines accions considera oportunes. Per exemple, descarregar una còpia legítima del binari substituït la modificació o se n'impedeix l'execució.

Validació de la llicència a un servidor de forma constant: Com el seu nom indica, aquesta tècnica consisteix a validar cada cert temps la llicència contra un servidor. No és el mecanisme més estable, ja que si el servidor està caigut pel motiu que sigui, un usuari legítim tindria dificultats per utilitzar-lo. En tot cas, dependrà de quan estrictes siguin les mesures aplicades pel desenvolupador. Tal com s'ha vist anteriorment, en aquest cas també es dificulta el procés de modificació del codi per tal d'executar el programa de forma fraudulenta, ja que caldria modificar el codi o prendre les accions oportunes per prèviament per evitar aquest control. Tanmateix, no ho impedeix.

Xifrar el codi: Consisteix a xifrar el codi i anar-lo desxifrant a mesura que es va executant. Tal com es pot veure, és només una trava, pel fet que la clau de desxifrat ha d'estar present i en text pla en l'equip si aquesta aplicació s'executa en l'equip de forma local.

Tokens hardware: Són dispositius de maquinari que depenent de la seva complexitat poden oferir més o menys dificultat en saltar-les. Per exemple, existeixen dispositius tals que tenen la capacitat de desxifrar el codi d'un programa xifrat al seu Interior i van carregant a la memòria del PC només la part és necessària per a la seva execució en cada moment. Per saltar aquest mecanisme caldria un estudi de l'electrònica del dispositiu extern.

Tècniques antidebuging: Existeixen funcions que es poden incloure en el codi del programa per detectar si s'adjunta algun depurador al procés de la mateixa aplicació, si s'introdueixen punts d'interrupció per maquinari examinant certs registres de la CPU o el temps que triga a executar un determinat codi (atenent que l'execució d'un debugger en fa augmentar aquest temps). Aquesta llista no és exhaustiva, no obstant això, mostra una idea general de com actuen aquestes proteccions.

Algunes conclusions que se'n poden extreure són que un generador de claus és més complexa de fer que modificar el binari en la majoria dels casos, tanmateix, és més efectiu. Tal com s'ha explicat en aquest mateix apartat, podria ser que en executar el programa es validés una signatura dels diferents arxius que el componen a algun servidor i, en detectar que hi ha una modificació, descarregués l'original una altra vegada o, fins i tot, una actualització del mateix programa podria invalidar aquestes modificacions o comprovacions de claus entre altres.

També, després d'analitzar el funcionament del sistema de validació de HDD Low Level Format Tool i el generador de claus per activar els mapes del navegador integrat en el Peugeot 208, es pot observar que, aparentment, és habitual l'ús de cadenes de text a les quals s'accedeix a una posició determinada per tal d'extreure'n un caràcter diverses vegades per formar o validar la clau.

Pel que fa a la planificació, la metodologia d'imposar-me petites fites a complir dins d'un determinat període de temps a set efectiva. Benauradament, no hi ha hagut sorpreses i tot ha transcorregut dins l'esperat.

En aquest aspecte, s'han realitzat algunes modificacions en el disseny del generador de claus del cas pràctic 2. Inicialment, es va desenvolupar en format Web (HTML + JavaScript) per permetre l'anàlisi del codi amb facilitat, tot i ser conscient que no era la solució més bona. Finalment, s'ha implementat en llenguatge C.

Altrament, després de la segona entrega, es va creure oportú ampliar aquest document i s'ha afegit un cas d'exemple addicional on s'analitza un generador de claus desenvolupat per tercers per tal d'entendre el mecanisme de registre d'uns mapes en un equip per un automòbil amb la intenció de donar-li més valor al TFG.

També, que el codi generat als diferents apartats, a banda d'adjuntar-lo a les entregues, era una bona idea posar-lo en un repositori GIT.

Pel que fa a les línies de treball futures, en aquest document s'han presentat aplicacions que duen a terme totes les validacions de les llicències localment. S'ha demostrat que aquest sistema és completament vulnerable. En un futur es considera abordar la seguretat de les aplicacions que es validen a través d'un servidor centralitzat, creant-ne un exemple i analitzant una aplicació del mercat. Això permetrà abordar els següents interrogants, així com altres qüestions que puguin sorgir:

- Com fa la petició al servidor a l'aplicació?
    - Les dades viatgen en text pla o xifrades?
        - En cas afirmatiu, quin sistema s'utilitza?
- El procés de validació és totalment opac a l'usuari, però el servidor és prou segur per evitar accessos no autoritzats?
- La informació que torna el servidor es rep en text pla?
- Es pot analitzar el que espera rebre l'aplicació del servidor per tal d'executar la rutina de registre?
    - En cas afirmatiu, es pot alterar perquè, malgrat que el servidor indiqui la que llicència no és vàlida, es registri igualment?

Actualment les aplicacions tendeixen, cada vegada més, a estar allotjades a un servidor i l'usuari en paga una llicència per el seu ús. D'aquesta manera no se'n pot estudiar el codi ni alterar-lo.

5. Glossari

A continuació es defineixen els termes i acrònims més rellevants utilitzats dins la Memòria.

Hacker: Una persona amb un coneixement elevat de sistemes informàtics i xarxes que fa servir aquest coneixement per explorar, modificar o millorar sistemes informàtics.

Hacker white hat: És un hacker que fa servir les seves habilitats informàtiques de manera ètica i legal. Solen ser professionals de la seguretat informàtica que treballen per millorar la seguretat dels sistemes informàtics.

Hacker grey hat: És una persona que fa servir habilitats informàtiques per dur a terme activitats que no són totalment legals ni totalment il·legals. Poden ser hackers que accedeixen a sistemes sense permís però amb intencions més benignes, com ara informar sobre vulnerabilitats o activisme social.

Hacker black hat: És un hacker que fa servir les seves habilitats amb finalitats malicioses o per a activitats il·lícites, com el frau, l'espionatge o l'alteració de sistemes informàtics amb intencions de causar un perjudici.

Hacktivistes: Són activistes informàtics que fan servir les seves habilitats per promoure causes socials o polítiques. Es podrien veure, en alguns casos, com a hackers amb moralitat Grey hat. Si bé comenten un il·lícit, ho fan amb la convicció de fer quelcom que ells creuen correcte.

Cracker: Una persona que manipula programari amb intencions malicioses, com ara trencar sistemes de seguretat, pirateria de programari o manipulació de codi per obtenir accés no autoritzat a sistemes informàtics.

Malware: Programari dissenyat específicament per causar danys o alterar de manera maliciosa els sistemes informàtics.

Cracking: Acció de trencar sistemes de seguretat informàtica amb la finalitat d'obtenir accés no autoritzat a sistemes, aplicacions o dades.

Keygening: Acció de crear generadors de claus de llicència o de codis d'activació per a programari amb finalitats il·legals.

Reversing: Procediment de desassemblatge i anàlisi del codi font o executable d'una aplicació amb l'objectiu de comprendre el seu funcionament intern.

Debugger: Una eina de programació utilitzada per a la identificació i correcció d'errors en el codi font d'un programa informàtic.

Copyright: Dret legal que protegeix els drets de propietat intel·lectual sobre obres creatives, com ara programari, música, literatura, etc., atorgant al creador o titular drets exclusius sobre la seva distribució i reproducció.

Formatatge: Procés d'eliminació de totes les dades d'un dispositiu de memòria, com ara un disc dur.

Debugging: Procés de localització i correcció d'errors (bugs) en el codi d'un programa informàtic.

String: Una seqüència de caràcters o símbols, com ara lletres, números o símbols especials, utilitzats per a representar text en programació informàtica.

Taula ASCII: És una taula de codis de caràcter estàndard que assigna un nombre únic a cada caràcter usat en els ordinadors i dispositius que utilitzen aquesta codificació de text.

Ensamblar: Procés d'escriure codi en llenguatge d'assemblador, que és posteriorment convertit en codi executable pel processador de l'ordinador.

Densensablar: Procés contrari a ensamblar: Obtenir un codi font proper a l'original a partir del propi executable ja compilat.

Hash: És una funció criptogràfica que transforma una entrada (o missatge) de qualsevol mida en una sortida de longitud fixa, coneguda com a valor hash o resum. Aquesta funció és determinista, és a dir, la mateixa entrada sempre produirà el mateix valor hash. Els valors hash s'utilitzen comunament per verificar la integritat de les dades, comparar informació de manera eficient i en aplicacions de seguretat, com ara l'emmagatzematge de contrasenyes i la signatura digital.

6. Bibliografia

En l'elaboració d'aquest Treball de fi de grau, no ha estat necessari consultar cap llibre, ja que el coneixement inicial de l'autor ha estat suficient per abordar el projecte, donat el seu caràcter pràctic. No obstant això, s'han utilitzat diverses fonts web per obtenir informació addicional i actualitzada sobre aspectes específics del tema tractat juntament amb alguns recursos. Aquestes consultes han permès complementar i enriquir el contingut del treball. Les referències bibliogràfiques corresponents a aquestes fonts en línia es troben degudament anotades a peu de pàgina per a la seva verificació i consulta.

7. Annexos

En els següents annexos es troba el material necessari per entendre el funcionament del programa x64dbg així com el codi font dels dos primers exemples.

7.1. ANNEX 1:

En aquest primer annex s'expliquen les funcionalitats utilitzades durant el desenvolupament dels casos pràctics en el "debugger" x64dbg:

Elements de la interfície:

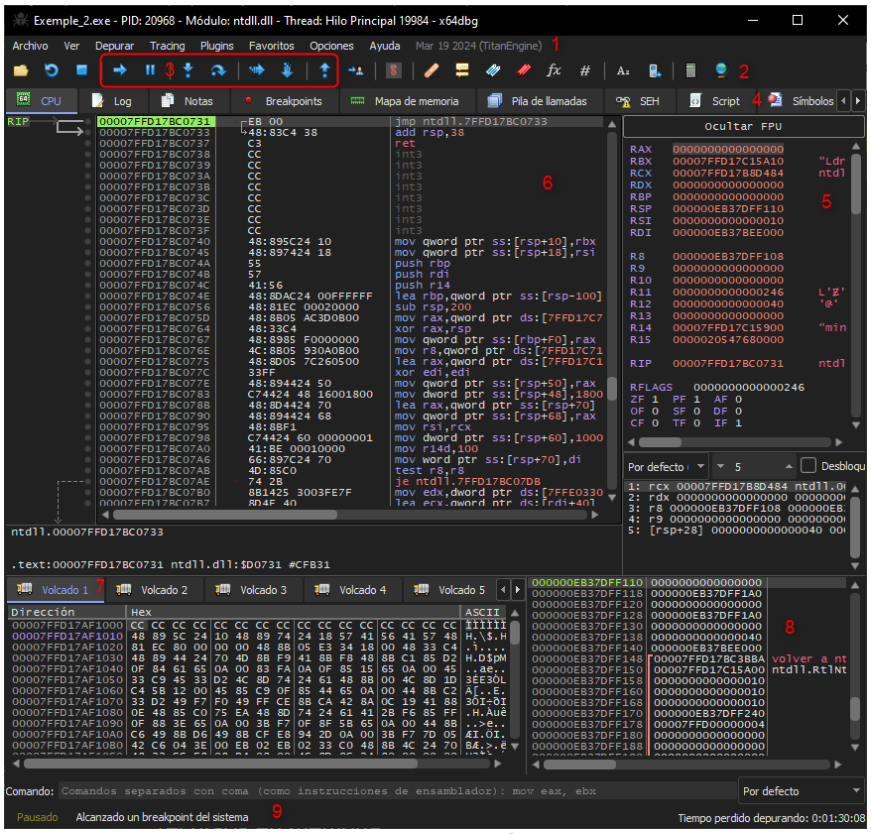

Figura 73: Annex 1: Interfície de x64dbg

1. Barra de menú: Conté opcions per obrir, desar i gestionar projectes, així com configuracions i opcions avançades del depurador.

2. Barra d'eines: Proporciona accessos directes a les funcions més utilitzades, com ara iniciar, aturar, posar en pausa la depuració, així com accedir a funcions de cerca i anàlisi.

3. Botons de control de navegació: Permeten navegar pel codi i les instruccions de forma ràpida i precisa.

4. Vistes: Diferents pestanyes on es pot controlar diferents aspectes de la depuració de forma individualitzada.

5. Vista de registre: Mostra els registres del processador i el seu estat actual durant l'execució del programa, incloent-hi registres generals com RAX, RBX, RCX, RDX, etc. (en la versió de 64 bits), així com registres de segment i de control.

6. Vista desensamblada: Mostra el codi assemblador del programa en execució, on es poden establir punts d'interrupció i seguir el flux d'execució, entre altres accions.

7. Vista de memòria: Permet examinar i modificar el contingut de la memòria del procés en temps real. Útil per a inspeccionar variables, estructures de dades i recursos en memòria.

8. Vista de pila: Mostra la pila actual durant l'execució del programa. Útil per a rastrejar l'execució de funcions i examinar els paràmetres i variables locals.

9. Barra d'estat: Mostra informació addicional sobre l'estat actual del depurador i el procés en execució, com ara l'estat de la depuració, el número de línia actual i missatges d'estat.

Funcionalitats utilitzades:

- Breackpoints:

Es pot afegir un “breackpoint” a un punt del programa en concret prement el botó dret del ratolí sobre qualsevol adreça de la següent manera:

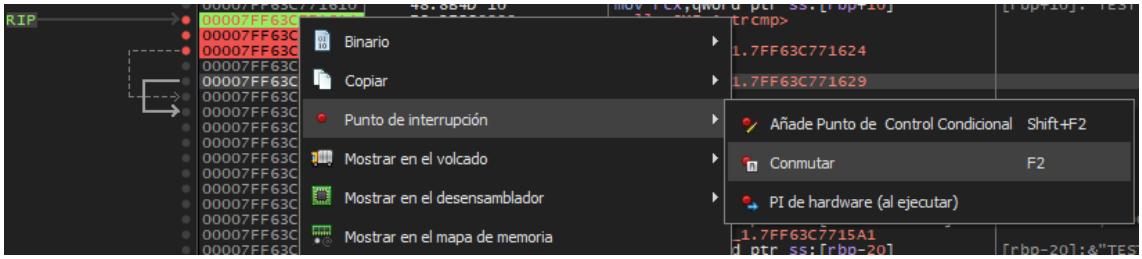

Figura 74: Annex 1: Punts d'interrupció amb x64dbg

Altrament, és possible consultar i administrar tots aquests punts d'interrupció que s'ha anat configurant des de la pestanya corresponent:

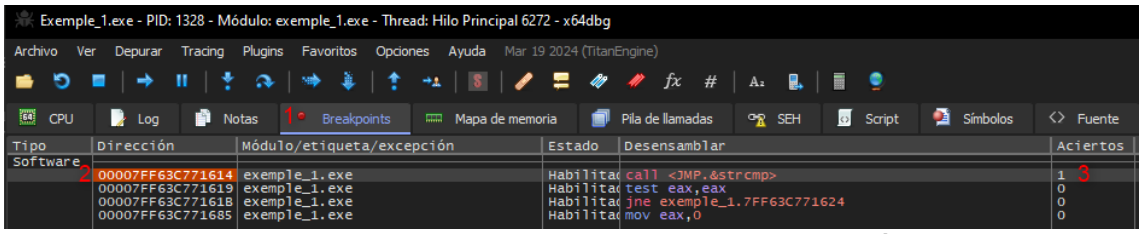

Figura 75: Annex 1: Detall dels punts d'interrupció

1. Pestanya on es poden administrar tots els "Breackpoints"
2. És possible eliminar un punt d'interrupció que ja no és necessari amb la tecla "Del" del teclat. Altrament, prement el botó dret del ratolí a sobre, hi ha algunes opcions que poden resultar útils.
3. S'indica el nombre de vegades que s'ha executat la instrucció en la qual hi ha configurat un punt d'interrupció.

- Ensamblar un valor nou:

És possible modificar una instrucció existent per comprovar el funcionament de l'executable amb aquesta modificació. Per fer-ho cal posicionar-se sobre la instrucció desitjada i prémer la barra espaiadora del teclat. També és possible fer-ho a través del menú contextual.

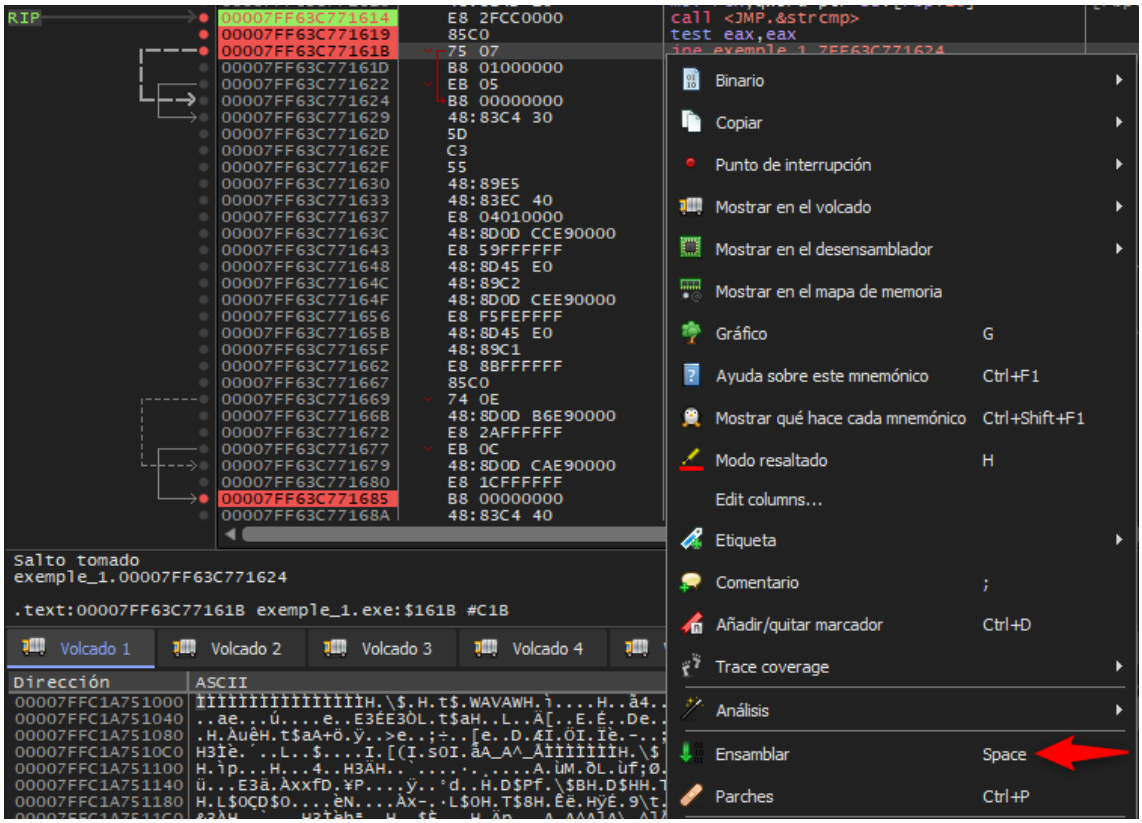

Figura 76: Annex 1: Ensambar una instrucció

Un cop feta la modificació, si es vol fer permanent, és possible desar una còpia del binari amb les modificacions aplicades. L'opció per fer-ho es troba al menú Arxiu i, seguidament, Apedaçar arxiu...

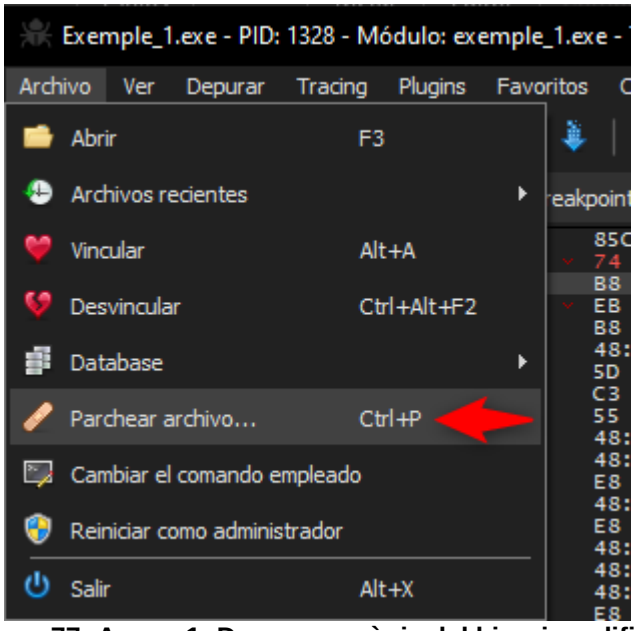

Figura 77: Annex 1: Desar una còpia del binari modificat

Es desplega un menú on es mostren totes les modificacions que s'han fet sobre el binari original i, finalment es desa la còpia modificada utilitzant l'opció: Aplicar el pedaç.

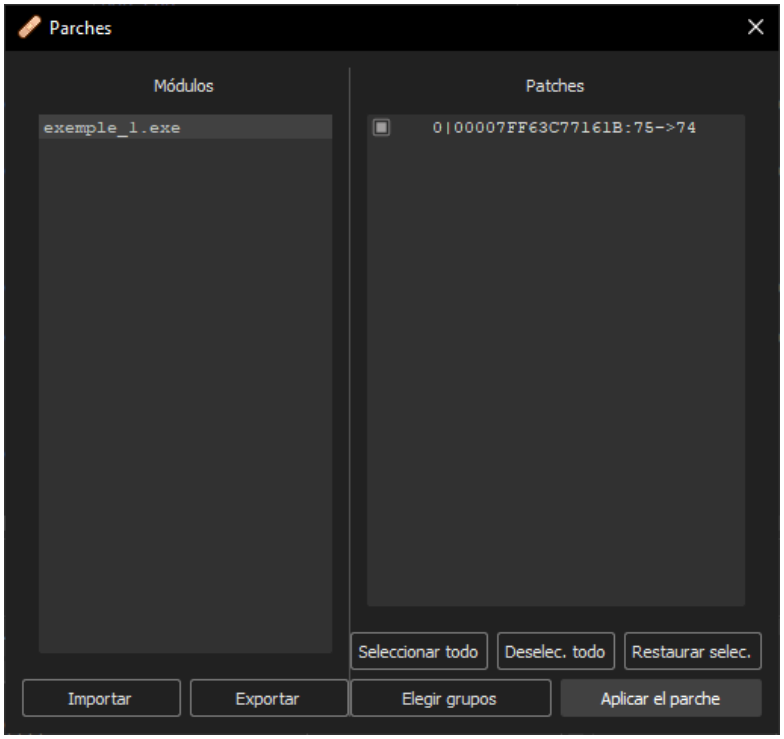

Figura 78: Annex 1: Detall del desat del binari modificat

- Crides entre mòduls, cerca de constants i cadenes de text:

Per trobar qualsevol d'aquests tres elements cal prémer el botó dret del ratolí damunt de la regió on es mostren totes les instruccions:

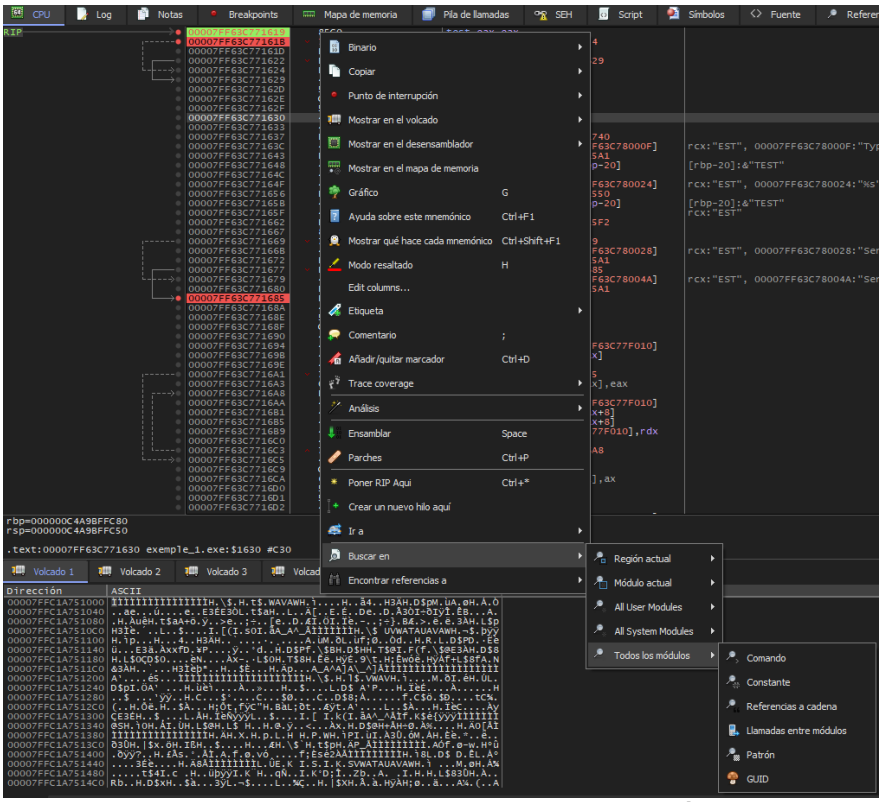

Figura 79: Annex 1: Cercar a dins del codi o la memòria a x64dbg

Tot seguit, cal triar en quina part del codi es volen buscar cadenes de text i quin element s'hi vol buscar en aquella regió en concret:

- Referències a cadena i crides entre mòduls: En ambdós casos es mostren els resultats en una pestanya nova. És possible utilitzar el cercador per acotar els resultats:

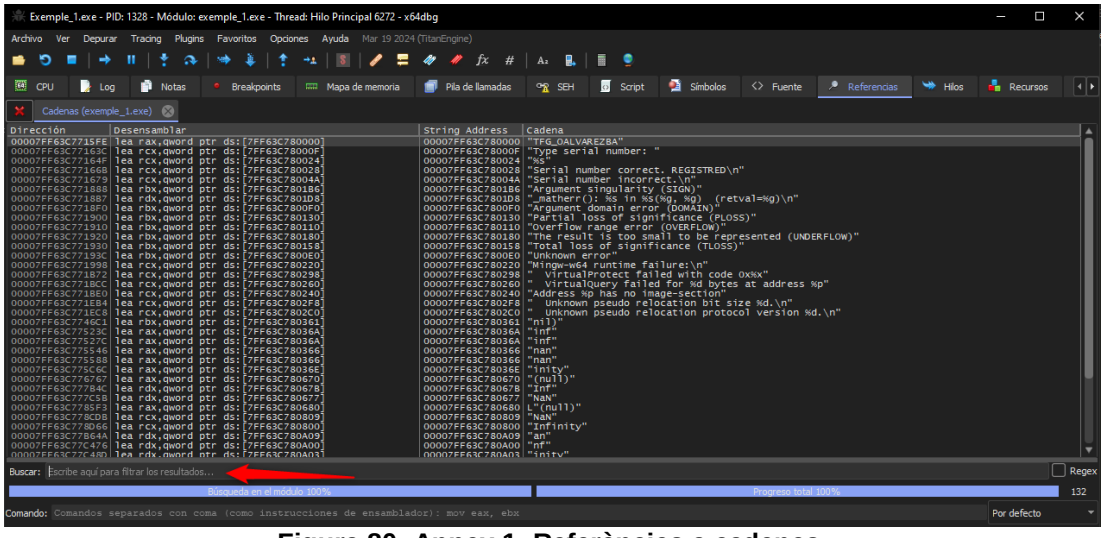

Figura 80: Annex 1: Referències a cadenes.

- Constants: L'opció de cerca de constants demana introduir el valor de la constant. Introduirem numèric en base decimal. El debugger x64dbg ja fa la traducció a base hexadecimal i mostra els resultats en forma de llista com els dos casos anteriors.

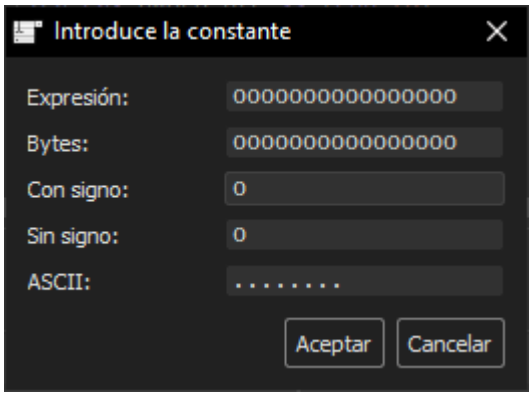

Figura 81: Annex 1: Cerques de constants

- Activar el mode gràfic: Es pot alternar entre el mode per defecte i el mode gràfic prement la tecla "G". Altrament, també és possible fer-ho des del menú contextual.

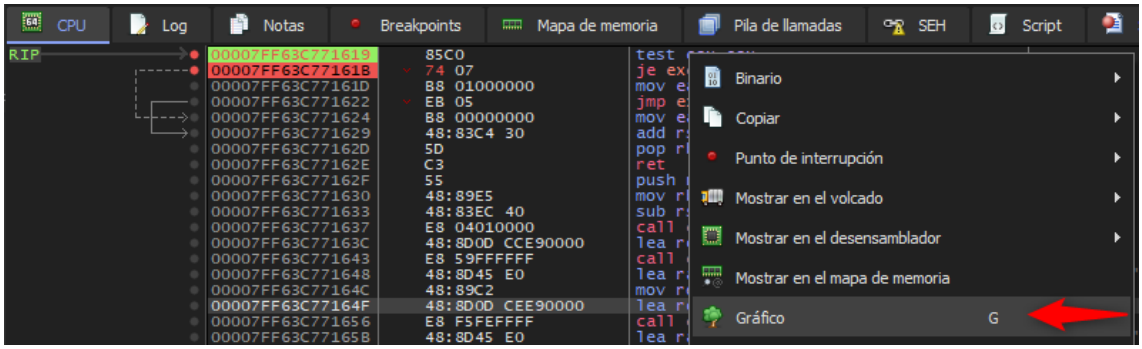

Figura 82: Annex 1: Mode gràfic

- Posicionar-se a una adreça en concret: Algunes vegades pot ser interessant posicionar-se a alguna adreça en concret per veure'n el contingut. Aquesta opció aplica tant per la memòria, com a la pila, com per posicionar-se a sobre l'adreça que apunta una instrucció. Es pot fer des del menú contextual amb l'opció Anar a i, seguidament, Expressió...

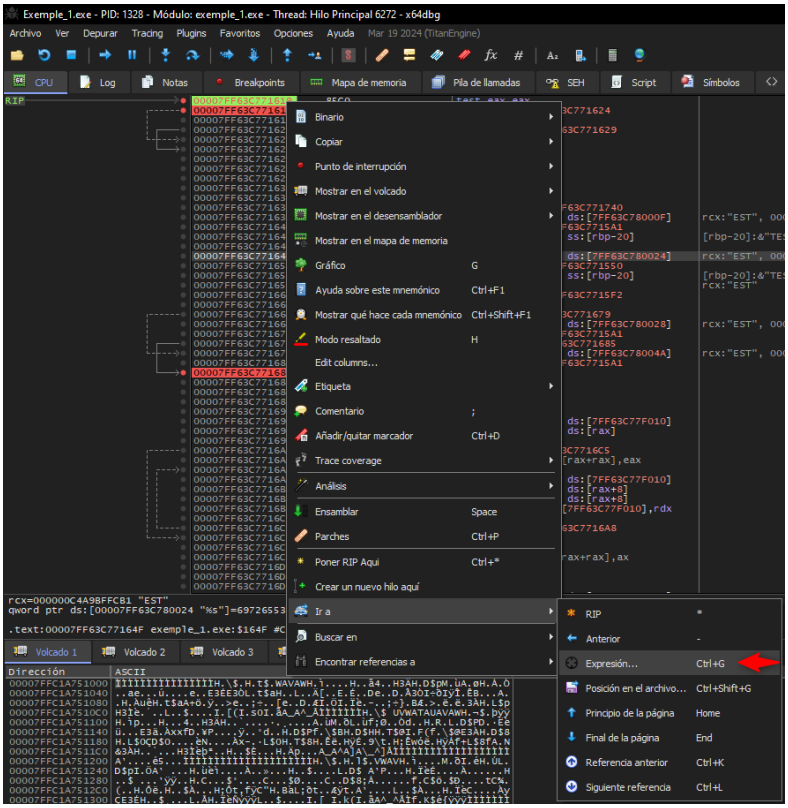

Figura 83: Annex 1: posicionar-se a una part del codi

7.2. ANNEX 2:

Codi en llenguatge C del primer exemple (Exemple_1.c):

```c
#include <stdio.h>
#include <string.h>

// Funció per validar el número de sèrie
int validarSerial(const char *serial) {
    // Número de sèrie predeterminat
    const char *serialCorrecte = "TFG_OALVAREZBA";

    // Comparar el número de sèrie ingressat amb el número de sèrie correcte
    if (strcmp(serial, serialCorrecte) == 0) {
        return 1; // Número de sèrie vàlid
    } else {
        return 0; // Número de sèrie no vàlid
    }
}

int main() {
    char serialUsuari[20];

    printf("Type serial number: ");
    scanf("%s", serialUsuari);

    // Cridar a la funció per validar el número de sèrie
    if (validarSerial(serialUsuari)) {
        printf("Serial number correct. REGISTRED\n");
    } else {
        printf("Serial number incorrect.\n");
    }

    return 0;
}
```

Figura 84: Annex 2: Codi font de Exemple_1.c

7.3. ANNEX 3:

Codi en llenguatge C del segon exemple (Exemple_2.c):

```c
#include <stdio.h>
#include <string.h>

// Funció per validar el número de sèrie
int validarSerial(const char *serial) {
    // Valor predefinit per a la suma dels caràcters ASCII del número de sèrie
    const int valorPredefinit = 1000;

    // Calcular la suma dels valors ASCII dels caràcters del número de sèrie
    int suma = 0;
    for (int i = 0; i < strlen(serial); i++) {
        suma += (int)serial[i]; // Sumar el valor ASCII de cada caràcter
    }

    // Comparar la suma calculada amb el valor predefinit
    if (suma == valorPredefinit) {
        return 1; // Número de sèrie vàlid
    } else {
        return 0; // Número de sèrie no vàlid
    }
}

int main() {
    char serialUsuari[20];

    printf("Type serial number: ");
    scanf("%s", serialUsuari);

    // Cridar a la funció per validar el número de sèrie
    if (validarSerial(serialUsuari)) {
        printf("Serial number correct. REGISTRED.\n");
    } else {
        printf("Serial number incorrect.\n");
    }

    return 0;
}
```

Figura 85: Annex 3: Codi font de Exemple_2.c