Ilmiy tadqiqotlar va ularning yechimlari jurnali
Aoka guvohnoma № 323759 ISSN: 3060-4818

PE FAYL FORMATIDAGI IMPORT VA EKSPORT JADVALLARI STRUKTURASINI BATAFSIL TAHLIL QILISH

Husniddinov Abbosjon Abrorjon o‘g‘li
Toshkent axborot texnologiyalari universiteti
, Kiberxavfsizlik fakulteti
4-bosqich talabasi, Toshkent, O‘zbekiston
E-mail: abbosjonhusniddinov0@gmail.com
+998905999605

Olimov Iskandar Salimboyevich
Senior Lecturer, Department of Cryptology
Tashkent University of Information
Technologies named after Muhammad al-Khwarizmi
E-mail: olimoviskandar89@gmail.com
https://doi.org/10.5281/zenodo.24194216

Annotatsiya

Ushbu ishda Windows operatsion tizimida bajariladigan fayllar uchun ishlatiladigan PE (Portable Executable) formatidagi import va eksport jadvallarining tuzilishi batafsil tahlil qilingan. PE faylning umumiy tuzilishi, RVA tushunchasi, import deskriptorlari, ILT va IAT jadvallari, bound va delay-load importlar, eksport jadvali hamda funksiyalarni qidirish algoritmi yoritilgan. Jadvallarni pefile kutubxonasi yordamida amalda tahlil qilish usullari keltirilgan va ularning zararli dasturlarni statik tahlil qilishdagi ahamiyati (shubhali API funksiyalar, siqilgan fayllar, imphash, IAT/EAT hooking) ko‘rsatilgan. Ishda haqiqiy PE fayllar (EXE va DLL) ustida o‘tkazilgan tajriba natijalari, imphashni qo‘lda hisoblash va shubhali API qidirishdagi noto‘g‘ri ijobiy (false positive) holatlar ham keltirilgan.

Kalit so‘zlar: PE format, import jadvali, eksport jadvali, IAT, RVA, imphash, zararli dasturlar tahlili, pefile.

Аннотация

В работе подробно проанализирована структура таблиц импорта и экспорта в формате PE (Portable Executable), который используется для исполняемых файлов в операционной системе Windows. Рассмотрены общая структура PE-файла, понятие RVA, дескрипторы импорта, таблицы ILT и IAT, связанный (bound) и отложенный (delay-load) импорт, таблица экспорта и алгоритм поиска функций. Показаны способы практического анализа таблиц с помощью библиотеки pefile, а также их значение для статического анализа вредоносных программ (подозрительные API-функции, упакованные файлы, imphash, перехват IAT/EAT). Приведены результаты эксперимента на реальных PE-файлах (EXE и DLL), ручной расчёт imphash и случаи ложных срабатываний при поиске подозрительных API.

Ключевые слова: формат PE, таблица импорта, таблица экспорта, IAT, RVA, imphash, анализ вредоносных программ, pefile.

Abstract

This paper presents a detailed analysis of the structure of import and export tables in the PE (Portable Executable) format, which is used for executable files in the Windows operating system. It covers the general layout of a PE file, the concept of RVA, import descriptors, the ILT and IAT tables, bound and delay-load imports, the export table, and the algorithm for looking up functions. Practical

methods of analyzing these tables with the pefile library are shown, along with their importance for static malware analysis (suspicious API functions, packed files, imphash, IAT/EAT hooking). The paper also reports an experiment on real PE files (EXE and DLL), a manual imphash calculation, and false-positive cases in suspicious-API detection.

Keywords: PE format, import table, export table, IAT, RVA, imphash, malware analysis, pefile.

KIRISH

Zamonaviy axborot xavfsizligi sohasida bajariladigan fayllarni tahlil qilish eng muhim yo‘nalishlardan biri bo‘lib qolmoqda. Microsoft Windows operatsion tizimida ishlaydigan deyarli barcha dasturlar, dinamik kutubxonalar va drayverlar PE (Portable Executable) formatida saqlanadi. Shu sababli zararli dasturni o‘rganuvchi mutaxassis, antivirus ishlab chiquvchisi yoki raqamli kriminalistika bilan shug‘ullanuvchi tahlilchi birinchi navbatda aynan shu format tuzilishini yaxshi bilishi kerak [1, 3].

PE faylning eng muhim qismlaridan biri import va eksport jadvallaridir. Import jadvali dastur ishlashi uchun qaysi tashqi kutubxonalardan (DLL) qaysi funksiyalar kerakligini bildiradi. Eksport jadvali esa kutubxona o‘zining qaysi funksiyalarini boshqa dasturlarga taqdim etishini ko‘rsatadi. Bu jadvallar faylni ishga tushirmasdan turib uning imkoniyatlari haqida xulosa chiqarishga imkon beradi: masalan, dastur tarmoq bilan ishlaydimi, fayllarni shifrlaydimi yoki boshqa jarayon xotirasiga yozadimi [1, 2].

Mavzuning dolzarbligi. Zararli dasturlar ko‘payib borayotgan sharoitda har bir namunani dinamik tahlil qilish (sandboxda ishga tushirish) vaqt va resurs jihatidan qimmat. Shu sababli statik tahlil, xususan import va eksport jadvallaridan olinadigan belgilar birlamchi saralashning asosiy vositalaridan biriga aylangan [3, 15]. Ayni paytda hujumchilar import jadvalini yashirish (dinamik hal qilish, API hashing) hamda qonuniy kutubxonalar niqobidan foydalanish (DLL proxying, IAT/EAT hooking) usullarini qo‘llaydi [10–12]. Bunday usullarni tushunish va aniqlash uchun jadvallar tuzilishini bayt darajasida bilish zarur.

Mavjud ishlar tahlili. PE formatining rasmiy tavsifi Microsoft hujjatida [1] va Pietrek maqolasida [2] bayon etilgan; zararli dasturlarni statik tahlil qilish amaliyoti Sikorski va Honig kitobida [3], Windows ichki tuzilishi esa [5] da yoritilgan. Import jadvaliga asoslangan imphash usuli Mandiant tomonidan taklif etilgan [6], hooking va xotira kriminalistikasi usullari [10, 11] da keltirilgan. So‘nggi yillarda PE sarlavhalari hamda import va eksport jadvallaridan olingan belgilar mashinaviy o‘rganish modellari uchun xususiyat sifatida ishlatilmoqda [13–15]. Mazkur ish shu bilimlarni o‘zbek tilida, nazariy bayon va haqiqiy fayllar ustidagi tekshiruv bilan birga keltirishga qaratilgan.

Ishning maqsadi — PE fayl formatidagi import va eksport jadvallarining tuzilishini batafsil o‘rganish, ularning ishlash mexanizmini tushuntirish hamda axborot xavfsizligi nuqtai nazaridan ahamiyatini ochib berish.

Maqsadga erishish uchun quyidagi **vazifalar** belgilandi:

– PE faylning umumiy tuzilishini va RVA tushunchasini o‘rganish;

– import jadvalining tarkibiy qismlarini (import deskriptorlari, ILT, IAT, bound va delay-load importlar) tahlil qilish;

– eksport jadvalining tuzilishi va funksiyani qidirish algoritmini ko‘rib chiqish;

– jadvallarni amalda pefile kutubxonasi yordamida o‘qish usulini ko‘rsatish;

– zararli dasturlarni tahlil qilishda import va eksport jadvallaridan foydalanish usullarini yoritish;

– haqiqiy PE fayllar (EXE va DLL) ustida tajriba o‘tkazib, nazariy bayonni va yozilgan skriptlarni tekshirish.

Tadqiqot usullari. Ish nazariy va amaliy qismlardan iborat. Nazariy qism Microsoft rasmiy hujjati [1], ilmiy-texnik adabiyotlar [2–5] va pefile hujjatlari [7] asosida tayyorlandi; barcha strukturalar nomi Microsoft hujjatlaridagi asl ko‘rinishida (masalan, IMAGE_IMPORT_DESCRIPTOR) berilgan. Amaliy

qismda Python 3.12 va pefile 2024.8.26 ishlatildi; tajriba ochiq manbali, zararsiz to‘rtta PE fayl ustida o‘tkazildi (4.6-band). Zararli namunalar ishlatilmadi.

Ishning amaliy natijalari. (1) Import va eksport jadvallari haqiqiy fayllarda qo‘lda (bayt darajasida) va pefile orqali o‘qilib, natijalar solishtirildi; (2) shubhali API qidiruvchi skript A/W variantlarini hisobga oladigan qilib ishlab chiqildi va zararsiz fayllarda noto‘g‘ri ijobiy natijalar ko‘rsatildi; (3) imphash ordinal importlarni hisobga olgan holda qo‘lda hisoblanib, pefile natijasi bilan tekshirildi.

1. PE FORMATI HAQIDA UMUMIY MA’LUMOT

1.1. PE formatining vazifasi va tarixi

PE (Portable Executable) formati Windows NT operatsion tizimi bilan birga paydo bo‘lgan va hozirgacha bajariladigan fayllar uchun asosiy format hisoblanadi. U .exe (dasturlar), .dll (dinamik kutubxonalar), .sys (drayverlar), .ocx, .scr kabi fayllarda ishlatiladi. Format UNIX tizimlaridagi COFF (Common Object File Format) asosida ishlab chiqilgan, shu sababli sarlavhalar orasida COFF nomi bilan atalgan qism saqlanib qolgan [1, 5].

Formatning nomidagi “portable” so‘zi uning turli protsessor arxitekturalarida (x86, x64, ARM) bir xil umumiy tuzilishga ega ekanini bildiradi. 32 bitli versiyasi PE32, 64 bitli versiyasi esa PE32+ deb ataladi. Ularning farqi asosan Optional Header hamda ayrim maydonlarning kengligida (32 yoki 64 bit) namoyon bo‘ladi [1].

PE fayl diskda saqlanayotgan holat bilan xotiraga yuklangan holat bir-biridan farq qiladi. Operatsion tizim yuklovchisi (loader) faylni xotiraga bevosita nusxalamaydi, balki uning bo‘limlarini (section) sarlavhalarda ko‘rsatilgan manzillarga joylashtiradi. Aynan shu sababli PE formatda fayl ichidagi offset va xotiradagi manzil o‘rtasida o‘zgartirish zarur bo‘ladi [5].

PE32 va PE32+ o‘rtasidagi asosiy farqlar quyidagilardan iborat:

– Magic maydoni: PE32 uchun 0x10B, PE32+ uchun 0x20B;

– ImageBase maydoni PE32 da 4 bayt, PE32+ da 8 bayt;

– Optional Header hajmi PE32 da standart holatda 224 bayt, PE32+ da 240 bayt; BaseOfData maydoni faqat PE32 da mavjud;

– ILT va IAT elementlari PE32 da 4 bayt, PE32+ da 8 bayt hajmga ega; ordinal import bayrog‘i PE32 da 31-bit, PE32+ da 63-bitdir.

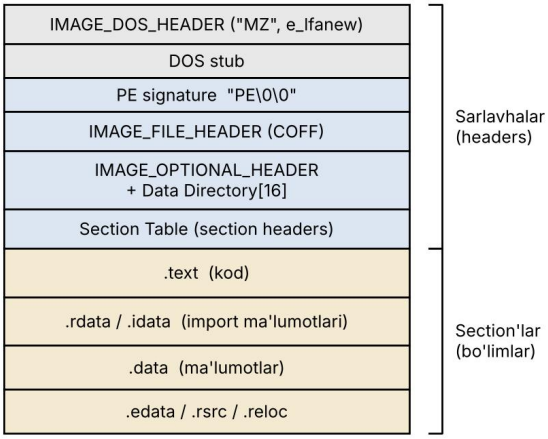
Bu farqlarni bilish import jadvalini o‘qiydigan dastur yozishda muhim: element kengligi noto‘g‘ri tanlansa, funksiyalar ro‘yxati buzilib o‘qiladi.

1.2. PE faylning umumiy tuzilishi

PE fayl ketma-ket joylashgan sarlavhalar va bo‘limlardan iborat (1-rasm). Fayl boshida eski MS-DOS dasturlari uchun qoldirilgan IMAGE_DOS_HEADER turadi. Uning birinchi ikki bayti MZ (0x5A4D) imzosi, 0x3C offsetdagi e_lfanew maydoni esa haqiqiy PE sarlavhasi qayerdan boshlanishini ko‘rsatadi. Undan keyin kichik DOS dasturi (DOS stub) keladi; u faylni DOS muhitida ishga tushirishga urinilsa, “This program cannot be run in DOS mode” xabarini chiqaradi [1, 2].

e_lfanew ko‘rsatgan joyda to‘rt baytli PE\0\0 (0x00004550) imzosi, undan keyin IMAGE_FILE_HEADER (COFF sarlavhasi) joylashadi. Bu sarlavhada protsessor turi (Machine), bo‘limlar soni (NumberOfSections), kompilyatsiya vaqti (TimeDateStamp), Optional Header hajmi va fayl xususiyatlari (Characteristics) saqlanadi [1].

Keyingi IMAGE_OPTIONAL_HEADER nomi chalkash: u aslida majburiy sarlavha, faqat obyekt fayllar uchun ixtiyoriy. Unda quyidagi muhim maydonlar bor: Magic (0x10B — PE32, 0x20B — PE32+), AddressOfEntryPoint (dastur kirish nuqtasi RVA), ImageBase (tavsiya etilgan yuklanish manzili), SectionAlignment va FileAlignment, SizeOfImage, Subsystem hamda DataDirectory massivi [1].



1-rasm. PE fayl tuzilishining umumiy sxemasi

1.3. Data Directory va bo‘limlar

DataDirectory — 16 ta elementdan iborat massiv. Har bir element ikki maydondan tarkib topadi: VirtualAddress (ma’lumot RVA manzili) va Size (hajmi). Aynan shu massiv orqali yuklovchi va tahlil dasturlari import, eksport, resurslar, relokatsiya va boshqa jadvallarni topadi. Eng muhim elementlar 1-jadvalda keltirilgan [1].

1-jadval. Data Directory ning asosiy elementlari

Indeks	Nomi	Vazifasi
0	Export Table	Faylning boshqalarga taqdim etadigan funksiyalari
1	Import Table	Fayl tashqaridan oladigan funksiyalar ro‘yxati
2	Resource Table	Piktogramma, satr, dialog kabi resurslar
5	Base Relocation	Boshqa manzilga yuklanganda tuzatiladigan joylar
6	Debug	Nosozliklarni tuzatish ma’lumotlari
9	TLS	Thread Local Storage ma’lumotlari
11	Bound Import	Oldindan bog‘langan (bound) importlar
12	IAT	Import Address Table manzili
13	Delay Import	Kechiktirib yuklanadigan importlar
14	CLR Runtime	Boshqariladigan (.NET) kod sarlavhasi

Sarlavhalardan keyin bo‘limlar jadvali (IMAGE_SECTION_HEADER massivi) keladi. Har bir bo‘lim sarlavhasi nom (Name, 8 bayt), xotiradagi hajm (VirtualSize), xotiradagi RVA (VirtualAddress), fayldagi hajm (SizeOfRawData), fayldagi offset (PointerToRawData) va xususiyatlar bayroqlarini (Characteristics) saqlaydi. Odatda uchraydigan bo‘limlar: .text (bajariladigan kod), .data (o‘zgaruvchilar), .rdata (faqat o‘qiladigan ma’lumotlar), .idata (import ma’lumotlari), .edata (eksport ma’lumotlari), .rsrc (resurslar) va .reloc (relokatsiyalar). Bo‘lim nomlari faqat odat bo‘lib, majburiy

emas: kompilyator yoki himoya vositasi ularni boshqacha atashi mumkin, shuning uchun jadvallarni topishda nomga emas, DataDirectory ga tayanish kerak. Shuni ta'kidlash kerakki, Microsoft kompilyatori (linkeri) import jadvalini odatda alohida .idata ga emas, .rdata bo'limi ichiga joylaydi (4.6-bandagi tajriba buni tasdiqladi); .idata esa asosan boshqa kompilyator va linkerlarda uchraydi [1, 3].

1.4. RVA tushunchasi va offsetga o'tkazish

PE formatda ko'p maydonlar manzilni RVA (Relative Virtual Address) ko'rinishida saqlaydi. RVA — faylning xotiradagi boshlang'ich manzili (ImageBase) ga nisbatan masofa. Xotiradagi to'liq manzil (VA) quyidagicha topiladi: VA = ImageBase + RVA [1].

Diskdagi faylni o'qiyotganda esa RVA ni fayl offsetiga aylantirish kerak bo'ladi. Buning uchun RVA qaysi bo'limga tushishini aniqlab, quyidagi formuladan foydalaniladi [1, 2]:

offset = RVA - Section.VirtualAddress + Section.PointerToRawData

Bu amal import va eksport jadvallarini qo'lda tahlil qilishda eng ko'p qo'llaniladigan hisob-kitobdir, chunki jadvallar ichidagi deyarli barcha ko'rsatkichlar RVA ko'rinishida beriladi.

2. IMPORT JADVALI TUZILISHI

2.1. Import mexanizmining zarurati

Windows dasturlari odatda barcha funksiyalarni o'zida saqlamaydi. Fayl bilan ishlash, oynalar yaratish, tarmoq ulanishlari kabi amallar tizim kutubxonalarida (kernel32.dll, user32.dll, ws2_32.dll va boshqalar) joylashgan. Dastur bu funksiyalarni chaqirishi uchun ularning xotiradagi aniq manzilini bilishi lozim. Lekin bu manzil Windows versiyasiga, kutubxona yangilanishiga va ASLR (Address Space Layout Randomization) himoyasiga qarab har safar o'zgarishi mumkin [5].

Muammo import jadvali orqali hal qilinadi: fayl tarkibida "menga ushbu DLL dan ushbu funksiyalar kerak" degan ro'yxat saqlanadi, dastur ishga tushirilganda esa yuklovchi bu funksiyalarning haqiqiy manzillarini topib, maxsus jadvalga yozib chiqadi. Dastur kodi funksiyani shu jadval orqali chaqiradi, shuning uchun kod o'zgarmasdan har qanday muhitda ishlayveradi [4, 5].

2.2. IMAGE_IMPORT_DESCRIPTOR strukturasi

DataDirectory[1] ko'rsatgan joyda IMAGE_IMPORT_DESCRIPTOR strukturalaridan iborat massiv turadi. Har bir struktura bitta DLL ga mos keladi va 20 bayt joy egallaydi. Massiv oxiri butunlay nol bilan to'ldirilgan struktura bilan belgilanadi. Strukturaning maydonlari 2-jadvalda keltirilgan [1].

2-jadval. IMAGE_IMPORT_DESCRIPTOR maydonlari

Maydon	Hajm	Ma'nosi
OriginalFirstThunk	4 bayt	Import Lookup Table (ILT) RVA manzili; funksiya nomlari yoki ordinallar ro'yxati
TimeDateStamp	4 bayt	0 — oddiy import; −1 — bound import qo'llangan
ForwarderChain	4 bayt	Forwarder zanjirining birinchi indeksi (odatda −1 yoki 0)
Name	4 bayt	DLL nomi yozilgan ASCII satr RVA manzili
FirstThunk	4 bayt	Import Address Table (IAT) RVA manzili

Ko'rinib turibdiki, bitta deskriptor bir vaqtning o'zida ikkita massivga ishora qiladi: OriginalFirstThunk — ILT ga, FirstThunk — IAT ga. Ularning vazifalari boshqa-boshqa, buni keyingi bandda ko'ramiz.

ForwarderChain maydoni bound import va yo'naltirish (forwarding) mexanizmi bilan bog'liq: u yo'naltirilgan funksiyalar zanjirining birinchi elementi indeksini bildiradi, yo'naltirish bo'lmaganda esa odatda 0 yoki −1 qiymatiga ega bo'ladi [1, 2]. 4.6-bandda tahlil qilingan barcha fayllarda import

deskriptorlarining TimeDateStamp va ForwarderChain maydonlari 0 ga teng bo‘lib chiqdi, ya’ni bu fayllarda bound import qo‘llanmagan.

2.3. ILT, IAT va IMAGE_IMPORT_BY_NAME

Import Lookup Table (ILT) — 32 bitli faylda 4 baytli, 64 bitli faylda 8 baytli elementlardan iborat massiv; oxiri nol element bilan tugaydi. Har bir elementning eng yuqori biti import turini bildiradi [1, 2]:

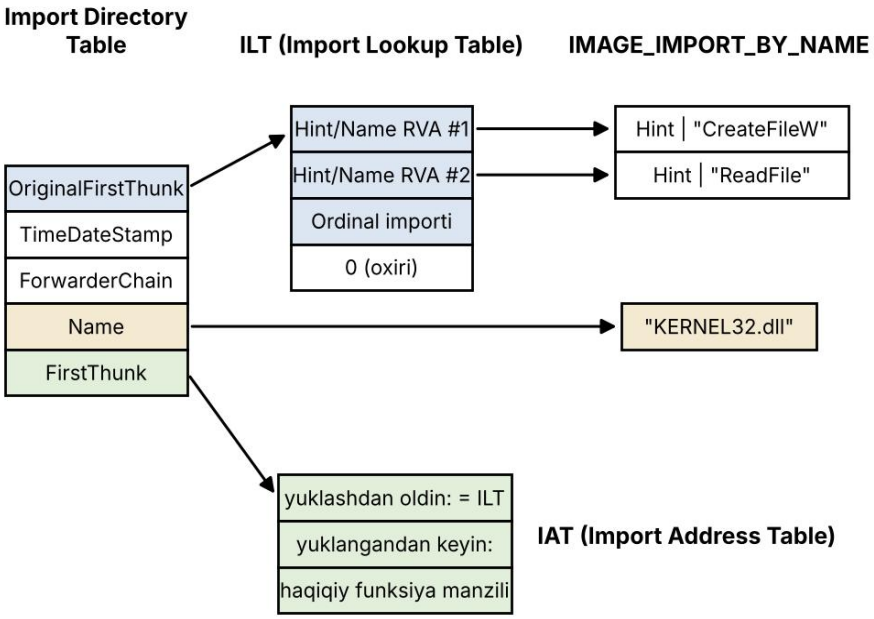
– **eng yuqori bit = 1** bo‘lsa, funksiya ordinal bo‘yicha import qilinadi, elementning quyi 16 biti ordinal raqamini bildiradi;

– **eng yuqori bit = 0** bo‘lsa, element IMAGE_IMPORT_BY_NAME strukturasiga RVA ishorasi hisoblanadi.

IMAGE_IMPORT_BY_NAME strukturasi ikki qismdan iborat: 2 baytli Hint (funksiyaning eksport nomlari jadvalidagi taxminiy indeksi) va oxiri nol bilan tugaydigan funksiya nomi (Name). Hint qiymati qidiruvni tezlashtirish uchun beriladi: yuklovchi avval shu indeksdagi nomni tekshiradi va mos kelmasa, nomni oddiy qidiruv bilan topadi [1, 2].

Import Address Table (IAT) fayl diskda saqlanayotganda ILT bilan bir xil mazmunga ega. Farq shundaki, ILT hech qachon o‘zgarmaydi, IAT esa dastur yuklanayotganda yuklovchi tomonidan funksiyalarning **haqiqiy manzillari** bilan to‘ldiriladi (2-rasm). Dastur kodidagi call dword ptr [IAT_element] kabi buyruqlar shu manzildan foydalanib, kerakli funksiyaga o‘tadi [1, 2].

Ordinal bo‘yicha import nom qidirishni talab qilmaganligi uchun tezroq ishlaydi va fayl hajmini kichraytiradi. Lekin ordinal raqamlari kutubxona versiyalari orasida o‘zgarishi mumkin, shuning uchun nom bo‘yicha import ishonchliroq hisoblanadi. Windows tizim DLL larida ordinallar versiyalar orasida bir xil saqlanmaydi, ularga tayanish tavsiya etilmaydi.



2-rasm. Import jadvalining tarkibiy qismlari orasidagi bog‘lanishlar

2.4. Yuklovchining import bilan ishlashi

Dastur ishga tushirilganda Windows yuklovchisi importlarni quyidagi tartibda hal qiladi [5]:

– PE faylni xotiraga joylashtiradi va DataDirectory[1] orqali import deskriptorlarini topadi;

– har bir deskriptor uchun Name maydonidagi DLL nomini o‘qiydi va kutubxonani yuklaydi (agar u allaqachon yuklangan bo‘lsa, qayta ishlatiladi);

– ILT elementlarini birma-bir o‘qib, har bir funksiyaning DLL eksport jadvalidagi manzilini topadi (nom yoki ordinal bo‘yicha);

– topilgan manzilni IAT dagi tegishli elementga yozadi;
– barcha importlar hal qilingandan so‘ng dastur kirish nuqtasiga boshqaruvni uzatadi.

Muhim jihati shundaki, yuklovchining o‘zi ham import qilinayotgan DLL ning **eksport jadvalini** o‘qiydi. Shuning uchun import va eksport jadvallari bir-biri bilan chambarchas bog‘liq: bir faylning importi boshqa fayl eksportiga ulanadi. Agar kerakli DLL yoki funksiya topilmasa, dastur ishga tushmaydi va foydalanuvchi “The procedure entry point could not be located” kabi xato xabarini ko‘radi [5].

2.5. Bound import

Oddiy importda har bir ishga tushirishda barcha funksiya manzillari qaytadan topiladi. Bu jarayonni tezlashtirish uchun bound import usuli ishlab chiqilgan. Bunda manzillar oldindan (odatda o‘rnatish paytida) hisoblanib IAT ga yozib qo‘yiladi, deskriptor TimeDateStamp maydoniga esa −1 qiymati qo‘yiladi. Bound import ma’lumotlari DataDirectory[11] orqali topiladigan IMAGE_BOUND_IMPORT_DESCRIPTOR strukturalarida saqlanadi: ularda DLL nomi va kutubxonaning kompilyatsiya vaqti (TimeDateStamp) yozilgan bo‘ladi [2, 5].

Yuklovchi bound importdagi vaqt belgisini haqiqiy DLL bilan solishtiradi. Mos kelsa, oldindan hisoblangan manzillar to‘g‘ri deb olinadi; mos kelmasa (masalan, DLL yangilangan bo‘lsa), importlar odatdagi usulda qayta hal qilinadi. ASLR joriy qilingandan keyin DLL manzillari tasodifiy bo‘lgani uchun bound import amalda deyarli foyda bermay qoldi va hozirgi dasturlarda kam uchraydi.

2.6. Delay-load import

Ayrim kutubxonalar dasturning faqat ma’lum funksiyalarida kerak bo‘ladi. Ularni ishga tushirish vaqtida yuklash resurs va vaqtni behuda sarflaydi. Shuning uchun delay-load usuli qo‘llanadi: kutubxona dastur boshlanishida emas, undagi funksiya **birinchi marta chaqirilganda** yuklanadi. Bunday importlar DataDirectory[13] orqali topiladigan IMAGE_DELAYLOAD_DESCRIPTOR strukturalarida tavsiflanadi. Ular DLL nomi, yuklangan modul tutqichi, delay IAT va delay INT (ImportNameTableRVA) kabi maydonlarni o‘z ichiga oladi [8].

Funksiya birinchi marta chaqirilganda kompilyator qo‘ygan yordamchi kod (__delayLoadHelper2) ishga tushadi, u LoadLibrary va GetProcAddress yordamida kutubxonani yuklaydi, manzilni delay IAT ga yozadi va chaqiruvni haqiqiy funksiyaga yo‘naltiradi. Keyingi chaqiruvlar to‘g‘ridan-to‘g‘ri ishlaydi. Tahlilchi uchun delay-load importlarni alohida ko‘rib chiqish muhim, chunki ular oddiy import jadvalida ko‘rinmaydi [8].

2.7. API Set (api-ms-win-*) importlari

Zamonaviy Windows tizimlarida, ayniqsa Windows 10 da, tizim funksiyalari ko‘pincha “API Set” deb ataluvchi virtual kutubxona nomlari orqali import qilinadi, masalan, api-ms-win-crt-runtime-l1-1-0.dll. Bunday nomga mos DLL fayl diskda alohida mavjud bo‘lmasligi mumkin: yuklovchi uni tizimning API Set sxemasi yordamida haqiqiy kutubxonaga (masalan, ucrtbase.dll yoki kernelbase.dll) yo‘naltiradi [9]. Tajribadagi cli-64.exe faylining 10 ta import deskriptoridan 8 tasi aynan api-ms-win-crt-* nomli (4.6-band). Tahlilchi uchun bu import jadvalidagi DLL nomi har doim diskdagi fayl nomi emasligini bildiradi: API Set nomini haqiqiy kutubxonaga moslashtirmasdan, funksiya qayerda amalga oshirilganini aniqlab bo‘lmaydi.

2.8. Boshqariladigan (.NET) fayllar

DataDirectory[14] (CLR Runtime Header) mavjud bo‘lgan .NET Framework fayllarida import jadvali deyarli bo‘sh: odatda faqat mscoree.dll dan _CorExeMain (EXE uchun) yoki _CorDllMain (DLL uchun) funksiyasi import qilinadi, chunki haqiqiy kod oraliq (IL) tilida bo‘lib, CLR tomonidan bajariladi [1, 3]. Shu sababli bunday fayllar uchun import jadvali va imphash deyarli ma’lumot bermaydi: ularning import ro‘yxati bir xil bo‘lgani uchun imphash ham bir xil chiqadi. Ularni tahlil qilishda CLR metama’lumotlari (TypeRef/MemberRef jadvallari) va P/Invoke (ImplMap) yozuvlari o‘rganiladi. Zamonaviy .NET (5 va undan yuqori) dasturlari esa odatda oddiy mashina kodidagi ishga

tushirgich bilan birga tarqatiladi va bunday ishga tushirgichning import jadvali oddiy PE fayldagi kabi bo‘ladi.

2.9. ELF formatidagi o‘xshash mexanizm (qiyosiy)

Linux va boshqa UNIX tizimlaridagi ELF formatida shunga o‘xshash mexanizm mavjud [17]. Taxminiy muvofiqliklar quyidagicha:

– **IAT** ↔ GOT (Global Offset Table): ikkalasi ham yuklovchi tomonidan haqiqiy manzillar bilan to‘ldiriladi;

– **ILT va IMAGE_IMPORT_BY_NAME** ↔ dinamik simvollar jadvali (.dynsym, .dynstr) va relokatsiyalar;

– **Eksport jadvali** ↔ .dynsym dagi aniqlangan (defined) simvollar;

– **Delay-load** ↔ PLT orqali kechiktirilgan bog‘lash (lazy binding).

Asosiy farq shundaki, PE da import qilinadigan funksiya har doim aniq bir DLL ga biriktirilgan (DLL → funksiyalar ro‘yxati), klassik ELF da esa simvollar kutubxonaga biriktirilmagan holda umumiy ro‘yxatda turadi, kutubxonalar nomi esa alohida yozuvlarda (DT_NEEDED) saqlanadi [17].

3. EKSPORT JADVALI TUZILISHI

3.1. IMAGE_EXPORT_DIRECTORY strukturasi

Eksport jadvali asosan DLL fayllarida uchraydi, lekin ayrim .exe fayllarda ham bo‘lishi mumkin. U DataDirectory[0] orqali topiladi va IMAGE_EXPORT_DIRECTORY strukturasi bilan boshlanadi. Bu struktura 40 bayt hajmga ega va 3-jadvalda keltirilgan maydonlardan iborat [1].

3-jadval. IMAGE_EXPORT_DIRECTORY maydonlari

Maydon	Hajm	Ma’nosi
Characteristics	4 bayt	Hozircha ishlatilmaydi, odatda 0
TimeDateStamp	4 bayt	Eksport jadvali yaratilgan vaqt
MajorVersion, MinorVersion	2+2 bayt	Versiya raqami (odatda 0)
Name	4 bayt	DLL nomi yozilgan ASCII satr RVA manzili
Base	4 bayt	Eksport ordinallarining boshlang‘ich qiymati (odatda 1)
NumberOfFunctions	4 bayt	Eksport manzillari jadvalidagi elementlar soni
NumberOfNames	4 bayt	Nomi bor eksport funksiyalar soni
AddressOfFunctions	4 bayt	Export Address Table (EAT) RVA manzili
AddressOfNames	4 bayt	Funksiya nomlari ishoralari massivi RVA manzili
AddressOfNameOrdinals	4 bayt	Nomlarga mos ordinal indekslari massivi RVA manzili

Bu yerda NumberOfFunctions va NumberOfNames qiymatlari bir xil bo‘lishi shart emas. Ayrim funksiyalar nomsiz, faqat ordinal orqali eksport qilinadi; bunday holatda NumberOfFunctions kattaroq bo‘ladi.

3.2. Uchta massiv va ularning bog‘lanishi

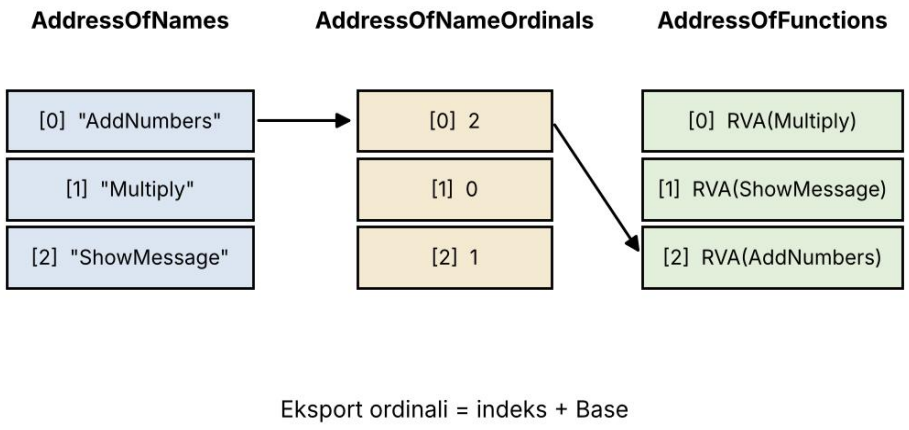
Eksport jadvalining asosini uchta massiv tashkil qiladi (3-rasm) [1]:

– **Export Address Table (EAT)** — AddressOfFunctions ko‘rsatgan massiv; har bir elementida eksport qilinayotgan funksiya yoki o‘zgaruvchining RVA manzili saqlanadi;

– **Name Pointer Table** — AddressOfNames ko‘rsatgan massiv; funksiya nomlari satrlariga RVA ishoralaridan iborat, nomlar alifbo (leksikografik) tartibida joylashtirilgan;

– **Ordinal Table** — AddressOfNameOrdinals ko‘rsatgan massiv; 2 baytli elementlardan iborat va har bir nomga mos EAT indeksini bildiradi.

Nomlar massivi va ordinallar massivi parallel ishlaydi: nomlar massividagi i-element ordinallar massividagi i-element bilan juft hisoblanadi. Ordinallar massividagi qiymat esa EAT dagi indeks bo‘ladi. Shuning uchun nomlar saralangan, EAT esa ordinal tartibida joylashgan [1].



3-rasm. Eksport jadvalida funksiyani nom bo‘yicha qidirish

3.3. Funksiyani qidirish algoritmi

Yuklovchi yoki GetProcAddress funksiyasi eksport qilingan funksiya manzilini topishda quyidagi ishlarni bajaradi [1, 2].

Nom bo‘yicha qidirish. Nomlar massivi alifbo tartibida saralanganligi uchun unda ikkilik qidiruv (binary search) qo‘llanadi. Kerakli nom i-o‘rinda topilsa, ordinallar massividan idx = AddressOfNameOrdinals[i] olinadi, undan keyin funksiya manzili RVA = AddressOfFunctions[idx] deb aniqlanadi. To‘liq manzil ImageBase + RVA ko‘rinishida hisoblanadi [1, 2].

Ordinal bo‘yicha qidirish. Bunda nomlar qidirilmaydi: berilgan ordinaldan Base ayriladi va natija EAT indeksi sifatida ishlatiladi: RVA = AddressOfFunctions[ordinal - Base]. Bu usul tezroq, lekin yuqorida aytilganidek, ordinallar versiyalar orasida o‘zgarishi mumkin [1, 2].

3.4. Forwarded export

Ba’zi funksiyalar kutubxonaning o‘zida emas, boshqa DLL da amalga oshirilgan bo‘ladi. Bunday holda EAT elementi funksiya manzilini emas, DLLNOMI.FunksiyaNomi ko‘rinishidagi satrga RVA ni saqlaydi (masalan, NTDLL.RtlAllocateHeap). Yuklovchi bu elementni ajratib olish uchun oddiy qoidadan foydalanadi: agar RVA eksport jadvali egallagan diapazon ichiga tushsa, u kod manzili emas, forward satri hisoblanadi. Shundan so‘ng ko‘rsatilgan DLL yuklanadi va funksiya shu yerdan izlanadi [1].

Forwarded export Microsoft tizim kutubxonalarida keng qo‘llaniladi: bu usul orqali dasturlarning eski kutubxona nomlariga bog‘liqligi saqlanib qoladi, haqiqiy amalga oshirish esa yangi yoki boshqa kutubxonaga ko‘chiriladi.

4. IMPORT VA EKSPORT JADVALLARINI AMALDA TAHLIL QILISH

4.1. Tahlil vositalari

PE fayl jadvallarini o‘rganish uchun turli vositalar mavjud. Grafik vositalardan PE-bear, CFF Explorer, PEview, Detect It Easy (DIE) keng qo‘llanadi: ular sarlavhalar, bo‘limlar, import va eksport jadvallarini daraxt ko‘rinishida ko‘rsatadi. Buyruq satri vositalaridan Microsoft Visual Studio tarkibidagi dumpbin /imports va dumpbin /exports buyruqlarini ko‘rsatish mumkin.

Dasturiy tahlil uchun eng qulay vositalardan biri — Python tilidagi pefile kutubxonasi. U PE faylni o‘qib, barcha sarlavha va jadvallarni Python obyektlari ko‘rinishida beradi. Quyida import va eksport jadvallarini chiqarishning oddiy namunasi keltirilgan [7].

```
import pefile

pe = pefile.PE("sample.exe")

# Import jadvali
if hasattr(pe, "DIRECTORY_ENTRY_IMPORT"):
    for entry in pe.DIRECTORY_ENTRY_IMPORT:
        print(entry.dll.decode())
        for imp in entry.imports:
            name = imp.name.decode() if imp.name else f"ord{imp.ordinal}"
            print(f"    {hex(imp.address)}  {name}")

# Eksport jadvali
if hasattr(pe, "DIRECTORY_ENTRY_EXPORT"):
    for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
        name = exp.name.decode() if exp.name else "-"
        va = pe.OPTIONAL_HEADER.ImageBase + exp.address
        print(hex(va), exp.ordinal, name)

# Import hash
print(pe.get_imphash())
```

1-listing. pefile yordamida import va eksport jadvallarini chiqarish

Skript birinchi navbatda DIRECTORY_ENTRY_IMPORT atributi mavjudligini tekshiradi, chunki import jadvali bo‘lmagan fayllar ham uchraydi. Har bir kutubxona uchun funksiyalar ro‘yxati va ularning IAT dagi manzillari chiqariladi. Eksport jadvali uchun (DIRECTORY_ENTRY_EXPORT) har bir funksiyaning to‘liq manzili, ordinali va nomi ko‘rsatiladi. Oxirida get_imphash() metodi import jadvalidan hisoblangan hash qiymatini qaytaradi, uning ahamiyati 5-bo‘limda tushuntiriladi.

4.2. Namunaviy natija

Oddiy konsol dastur uchun import jadvalining namunaviy ko‘rinishi 4-jadvalda berilgan. Bu yerda faqat tushunish uchun soddalashtirilgan misol keltirilgan; haqiqiy faylda funksiyalar soni ko‘proq bo‘ladi.

4-jadval. Namunaviy import jadvali (illyustrativ misol)

DLL	Funksiya	Vazifasi
KERNEL32.dll	CreateFileW	Fayl ochish yoki yaratish
KERNEL32.dll	ReadFile	Fayldan o‘qish
KERNEL32.dll	ExitProcess	Jarayonni tugatish
USER32.dll	MessageBoxW	Xabar oynasini ko‘rsatish

DLL	Funksiya	Vazifasi
ADVAPI32.dll	RegOpenKeyExW	Reestr kalitini ochish

Bunday jadvaldan dastur fayllar va reestr bilan ishlashini hamda foydalanuvchiga oyna ko‘rsatishini fayl kodini bajarmasdan bilib olish mumkin. Amaliy mashg‘ulotda o‘z kompyuteringizdagi biror dastur (notepad.exe) yoki kutubxona (kernel32.dll) ustida shu skriptni ishga tushirib, natijani o‘zingiz tekshirib ko‘rishingiz mumkin. Haqiqiy fayllar ustida olingan natijalar 4.6-bandda keltirilgan.

4.3. Qo‘lda tahlil bosqichlari

Vositalar qanday ishlashini tushunish uchun jadvallarni qo‘lda hex-muharrirda o‘qish tartibini bilish foydali. Import jadvalini o‘qishning umumiy bosqichlari:

– 0x3C offsetdan e_lfanew ni o‘qib, PE sarlavhasini topish;

– Optional Header ichidagi DataDirectory[1] dan import jadvali RVA manzili va hajmini olish;

– bo‘limlar jadvalidan shu RVA qaysi bo‘limga tushishini aniqlab, uni 1.4-bandagi formula bilan offsetga aylantirish;

– har 20 baytlik deskriptorni o‘qib, Name orqali DLL nomini, OriginalFirstThunk orqali funksiyalar ro‘yxatini topish;

– nol bilan to‘ldirilgan deskriptorga duch kelganda to‘xtash.

Eksport jadvali ham shunga o‘xshash o‘qiladi: DataDirectory[0] dan IMAGE_EXPORT_DIRECTORY topiladi, undan uch massiv RVA manzillari olinadi va 3.3-bandagi algoritm bo‘yicha nomlar va manzillar juftlanadi.

4.3.1. Hisoblash misoli

Quyidagi misol tushunish uchun ixtiyoriy tanlangan qiymatlar bilan berilgan (haqiqiy fayldan olinmagan). Faraz qilaylik, .idata bo‘limining VirtualAddress qiymati 0x3000, PointerToRawData qiymati esa 0x1800 bo‘lsin, DataDirectory[1] dagi import jadvali RVA manzili 0x3040 bo‘lsin. U holda jadvalning fayldagi offseti quyidagicha topiladi:

offset = 0x3040 - 0x3000 + 0x1800 = 0x1840

Hex-muharrirda 0x1840 offsetdan boshlab birinchi 20 baytlik deskriptor o‘qiladi. Uning Name maydonida 0x30A0 qiymati turgan bo‘lsa, DLL nomi 0x30A0 - 0x3000 + 0x1800 = 0x18A0 offsetda bo‘ladi va u yerda “KERNEL32.dll” satri o‘qiladi. Xuddi shu usul bilan OriginalFirstThunk ham offsetga aylantirilib, funksiya nomlari (IMAGE_IMPORT_BY_NAME) topiladi. Natijada har bir qadamda bir xil formula qo‘llanishini ko‘rish mumkin: avval RVA qaysi bo‘limga tegishli ekani aniqlanadi, so‘ng offset hisoblanadi.

Eksport jadvalida ham xuddi shunday. Faraz qilaylik, AddressOfNames massividagi birinchi element 0x30F0 ga teng. Uning offseti 0x30F0 - 0x3000 + 0x1800 = 0x18F0 bo‘ladi va shu joydagi nol bilan tugaydigan satr funksiya nomi hisoblanadi. Bu nomga mos ordinal indeksi ordinallar massividan olinadi, undan keyin EAT dagi RVA yana shu formula bilan offsetga aylantiriladi.

Eslatma: misolda soddalik uchun import va eksport jadvallari bir xil bo‘limda joylashgan deb olindi. Amalda ular odatda turli bo‘limlarda (masalan, .rdata va .edata) bo‘ladi, shuning uchun har bir RVA uchun mos bo‘lim alohida aniqlanadi [1]. Haqiqiy fayl misoli 4.6-bandda keltirilgan.

4.4. Shubhali importlarni avtomatik aniqlash

Ko‘p sonli fayllarni tahlil qilganda jadvallarni qo‘lda ko‘rib chiqish samarasiz. Quyidagi skript 6-jadvalda keltirilgan funksiya guruhlaridan ayrimlarini import ro‘yxatidan qidiradi, shuningdek importlar soni kam va bo‘limlar entropiyasi yuqori bo‘lsa, faylni siqilgan bo‘lishi mumkin deb belgilaydi.

```
import sys, pefile

SUSPICIOUS = {
    "OpenProcess", "VirtualAllocEx", "WriteProcessMemory",
    "CreateRemoteThread", "SetWindowsHookEx", "GetAsyncKeyState",
```

```
    "URLDownloadToFile", "InternetOpenUrl", "CryptEncrypt",
    "CreateService", "RegSetValueEx", "IsDebuggerPresent",
}

def normalize(name):
    # A (ANSI) va W (Unicode) qo‘shimchalarini olib tashlash
    if name[-1:] in ("A", "W") and name[:-1] in SUSPICIOUS:
        return name[:-1]
    return name

def analyze(path):
    pe = pefile.PE(path)
    found, total = [], 0
    for entry in getattr(pe, "DIRECTORY_ENTRY_IMPORT", []):
        for imp in entry.imports:
            total += 1
            name = imp.name.decode() if imp.name else ""
            if normalize(name) in SUSPICIOUS:
                found.append(entry.dll.decode() + "!" + name)
    entropy = max(s.get_entropy() for s in pe.sections)
    packed = total < 10 and entropy > 7.0
    return total, found, packed

total, found, packed = analyze(sys.argv[1])
print("Jami import:", total)
print("Shubhali:", ", ".join(found) if found else "topilmadi")
if packed:
    print("Ogohlantirish: fayl siqilgan bo‘lishi mumkin")
```

2-listing. Shubhali import funksiyalarini qidirish

Skript funksiya nomlaridagi A (ANSI) va W (Unicode) qo‘shimchalarini normallashtiradi (masalan, URLDownloadToFileA va URLDownloadToFileW bitta funksiya sifatida qaraladi); aks holda faqat bitta variantni ishlatgan namuna e’tibordan chetda qolardi. getattr(pe, "DIRECTORY_ENTRY_IMPORT", []) chaqiruvi import jadvali bo‘lmagan fayllarda ham xatosiz ishlaydi. Siqilganlik belgisi sifatida import sonining kamligi (10 dan kam) va yuqori entropiya (7,0 dan yuqori) birgalikda tekshiriladi; bu chegaralar qat’iy qoida emas, balki amaliyotda ishlatiladigan taxminiy qiymatlar (heuristika) bo‘lib, namunalar to‘plamida sinab ko‘rilishi kerak. Skript natijasi yakuniy xulosa emas: 4.6-bandda ko‘rsatilganidek, zararsiz fayllarda ham IsDebuggerPresent yoki OpenProcess kabi funksiyalar uchraydi, shuning uchun haqiqiy tizimlarda bunday qoidalar bir necha belgi ballari bilan birga baholanadi.

4.5. Imphash ni qo‘lda hisoblash

Imphash qanday hosil bo‘lishini tushunish uchun uni qo‘lda hisoblab ko‘rish foydali. Quyidagi funksiya 5.3-bandda bayon etilgan qoidani amalga oshiradi.

```
import hashlib, pefile
from pefile import ordlookup

def my_imphash(path):
    pe = pefile.PE(path)
    items = []
```

```
    for entry in getattr(pe, "DIRECTORY_ENTRY_IMPORT", []):
        dll = entry.dll.decode().lower()
        base, _, ext = dll.rpartition(".")
        if ext in ("ocx", "sys", "dll"):   # kengaytma
            dll = base
        for imp in entry.imports:
            if imp.name:
                func = imp.name.decode()
            else:                          # ordinal
                lib = entry.dll.lower()
                func = ordlookup.ordLookup(lib, imp.ordinal, True)
                if isinstance(func, bytes):
                    func = func.decode()
            items.append(dll + "." + func.lower())
    return hashlib.md5(",".join(items).encode()).hexdigest()
```

3-listing. Imphash ni qo‘lda hisoblash

Funksiya imphash hisoblash qoidasini to‘liq takrorlaydi: DLL nomi kichik harflarga o‘tkaziladi va faqat .dll, .ocx, .sys kengaytmalari olib tashlanadi; ordinal bo‘yicha import qilingan funksiyalar uchun nom pefile.ordlookup moduli orqali aniqlanadi (ws2_32, wsock32, oleaut32 kutubxonalari uchun ma’lum funksiya nomlari, boshqa hollarda ord7 kabi shartli nom). Agar faqat nomli importlar hisobga olinsa, ordinal importi bor fayllarda natija get_imphash() natijasidan farq qiladi [6, 7]. 4.6-bandda to‘rtta fayl uchun my_imphash va get_imphash() natijalari bir xil chiqqani ko‘rsatilgan. Hisoblangan qiymatni VirusTotal kabi xizmatlarda qidirib, shu importlar to‘plamiga ega boshqa fayllarni topish mumkin.

4.6. Haqiqiy fayllar ustida tajriba

Nazariy bayonni va yuqoridagi skriptlarni tekshirish uchun ochiq manbali, zararsiz to‘rtta PE fayl tanlandi: setuptools 84.0.0 paketidagi cli-64.exe (PE32+) va cli-32.exe (PE32) ishga tushirgichlari hamda PyPI dagi Windows (win_amd64) wheel paketlaridan olingan markupsafe 3.0.4 (_speedups.cp312-win_amd64.pyd) va psutil 7.2.2 (_psutil_windows.pyd) kengaytma modullari. .pyd fayllari aslida DLL bo‘lib, eksport jadvaliga ega. Tahlil Python 3.12 va pefile 2024.8.26 da o‘tkazildi. Importlarni bayt darajasida o‘qish uchun yozilgan skript 4-listingda keltirilgan; u PE32 va PE32+ fayllarni ham qayta ishlaydi.

```
import struct, sys

def rva2off(rva, secs):
    for va, vsize, raw in secs:
        if va <= rva < va + max(vsize, 1):
            return rva - va + raw
    raise ValueError(hex(rva))

d = open(sys.argv[1], "rb").read()
pe = struct.unpack_from("<I", d, 0x3C)[0]        # e_lfanew
nsec, = struct.unpack_from("<H", d, pe + 6)      # bo‘limlar soni
optsize, = struct.unpack_from("<H", d, pe + 20)  # Optional Header
opt = pe + 24
plus = struct.unpack_from("<H", d, opt)[0] == 0x20B  # PE32+ mi?
dd = opt + (112 if plus else 96)                 # DataDirectory
imp_rva, _ = struct.unpack_from("<II", d, dd + 8)    # indeks 1
secs = []
```

```
    for i in range(nsec):
        o = opt + optsize + 40 * i
        vsize, va, _, raw = struct.unpack_from("<4I", d, o + 8)
        secs.append((va, vsize, raw))
    off = rva2off(imp_rva, secs)
    fmt, step = ("<Q", 8) if plus else ("<I", 4)
    while True:
        oft, _, _, name, ft = struct.unpack_from("<5I", d, off)
        if not (oft or name or ft):                # nol deskriptor
            break
        t, n = rva2off(oft or ft, secs), 0
        while struct.unpack_from(fmt, d, t + n * step)[0]:
            n += 1
        no = rva2off(name, secs)
        print(d[no:d.index(b"\0", no)].decode(), n)
        off += 20
```

4-listing. Import jadvalini bayt darajasida o‘qish (pefile siz)

To‘rtta faylning umumiy ko‘rsatkichlari 5-jadvalda berilgan. Qo‘lda o‘qish skripti har bir fayl uchun pefile bilan bir xil DLL va import sonini berdi (masalan, cli-64.exe uchun 10 ta DLL va 64 ta import).

5-jadval. Tajriba uchun tanlangan PE fayllar va ularning import/eksport ko‘rsatkichlari

Fayl	Turi	DLL / import	Eksport	Imphash (16 belgi)
cli-64.exe	PE32+, EXE	10 / 64	yo‘q	77d2a6fffe40a245
cli-32.exe	PE32, EXE	10 / 61	yo‘q	e38062877caac655
_speedups.cp312-win_amd64.pyd	PE32+, DLL	4 / 19	PyInit__speedups	742935dedc475bd9
_psutil_windows.pyd	PE32+, DLL	15 / 182	PyInit__psutil_windows	ba3f3e54980963c2

Tajriba natijalari quyidagicha:

– **Import jadvalining joylashuvi.** cli-64.exe da DataDirectory[1] import jadvalini RVA 0x3A04 (hajmi 220 bayt, ya’ni 10 ta deskriptor va oxirgi nol deskriptor) deb ko‘rsatadi. Bu RVA .rdata bo‘limiga (VirtualAddress = 0x3000, PointerToRawData = 0x1C00) tushadi; faylda .idata bo‘limi yo‘q. Demak, Microsoft linkeri import jadvalini .rdata ichiga joylagan (1.3-band).

– **Qo‘lda hisoblash.** offset = 0x3A04 − 0x3000 + 0x1C00 = 0x2604. Shu offsetdan boshlab 10 ta deskriptor o‘qildi, 11-si nol. Birinchi deskriptorning Name maydoni 0x3DE2 bo‘lib, u 0x3DE2 − 0x3000 + 0x1C00 = 0x29E2 offsetda “KERNEL32.dll” satriga olib keldi; OriginalFirstThunk = 0x3AE0 dan 23 ta 8 baytli element o‘qildi, bu pefile natijasiga (23 ta funksiya) mos. FirstThunk = 0x3000, ya’ni IAT .rdata bo‘limining boshida turibdi.

– **API Set.** cli-64.exe ning 10 ta DLL idan 8 tasi api-ms-win-crt-* nomli, qolgani KERNEL32.dll va VCRUNTIME140.dll (2.7-band).

– **PE32 va PE32+.** Bir dasturning 32 va 64 bitli yig‘malari import soni (61 va 64) hamda imphash qiymatlari bilan farq qiladi: imphash bir manba kodining turli arxitektura yig‘malarini bir oilaga

birlashtirmaydi. Farq, masalan, _except_handler4_common (32 bit) va __C_specific_handler (64 bit) importlaridan kelib chiqadi.

– **Eksport jadvali.** markupsafe modulida DataDirectory[0] RVA 0x3800 ni ko‘rsatadi; Base = 1, NumberOfFunctions = NumberOfNames = 1, yagona eksport — PyInit__speedups (RVA 0x16D0). Python bu modulni aynan PyInit_<modul nomi> funksiyasi orqali ishga tushiradi, shuning uchun bunday .pyd fayllarining eksport jadvali bitta yozuvdan iborat bo‘ladi.

– **Noto‘g‘ri ijobiy natijalar.** 2-listing skripti cli-64.exe va cli-32.exe da IsDebuggerPresent ni, _psutil_windows.pyd da esa OpenProcess va IsDebuggerPresent ni “shubhali” deb topdi (markupsafe da hech narsa topilmadi). Bu fayllar zararsiz va ochiq manbali: IsDebuggerPresent Microsoft C/C++ ish vaqti kutubxonasi bilan bog‘liq import bo‘lib, OpenProcess esa psutil ning jarayonlar haqidagi ma’lumotni olish vazifasi uchun kerak. Demak, yakka funksiya mavjudligi zararli niyatning isboti emas (5.1-band).

– **Imphash tekshiruvi.** 3-listingdagi my_imphash funksiyasi to‘rtta faylning hammasi uchun pefile.get_imphash() bilan bir xil qiymat berdi.

Tajriba cheklovlari: tanlangan fayllarning hech birida bound import, delay-load import yoki ordinal bo‘yicha import uchramadi, shuning uchun ordinallarni qayta ishlovchi qism pefile manba kodiga muvofiq yozildi, ammo amalda sinalmadi. Zararli namunalar ishlatilmagani uchun aniqlash samaradorligi (to‘g‘ri va noto‘g‘ri ijobiy natijalar ulushi) statistik baholanmadi.

5. IMPORT VA EKSPORT JADVALLARINING XAVFSIZLIK NUQTAI NAZARIDAN AHAMIYATI

5.1. Statik tahlilda import jadvalining qiymati

Zararli dasturni tahlil qilishda import jadvali birinchi navbatda ko‘riladigan ma’lumotlardan biri hisoblanadi. Sababi, dasturning har qanday muhim amali (fayl yaratish, tarmoqqa ulanish, jarayon yaratish, shifrlash) oxir-oqibat operatsion tizim funksiyalariga murojaat orqali bajariladi. Import ro‘yxati dasturning **qanday imkoniyatlarga ega ekanini** ko‘rsatadi, ammo uning aynan nima qilishini isbotlamaydi: ko‘p funksiyalar oddiy dasturlarda ham ishlatiladi [3, 16].

6-jadvalda amalda shubha uyg‘otadigan funksiyalarning ayrim guruhlari keltirilgan. Yakka funksiyaning bo‘lishi hali hech narsani isbotlamaydi; muhimi ularning birgalikda uchrashi. Bunga 4.6-bandda misol keltirilgan: zararsiz cli-64.exe da IsDebuggerPresent, psutil modulida esa OpenProcess topildi.

6-jadval. Ko‘p uchraydigan shubhali API funksiyalar guruhlari

Guruh	Funksiyalar	Mumkin bo‘lgan maqsad
Jarayon xotirasi bilan ishlash	OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread	Boshqa jarayonga kod kiritish (process injection)
Tarmoq	InternetOpenW, InternetOpenUrlW, URLDownloadToFileW, connect, send	Fayl yuklab olish, boshqaruv serveriga ulanish
Shifrlash	CryptAcquireContextW, CryptEncrypt, CryptGenKey	Ma’lumotlarni shifrlash (masalan, shifrlovchi dasturlarda)
Klaviatura va oyna	SetWindowsHookExW, GetAsyncKeyState, GetForegroundWindow	Tugmalarni kuzatish (keylogger)

Guruh	Funksiyalar	Mumkin bo‘lgan maqsad
Reestr va xizmatlar	RegSetValueExW, CreateServiceW	Tizimga o‘rnashib olish (persistence)
Tahlildan himoya	IsDebuggerPresent, CheckRemoteDebuggerPresent	Tahlilchi dasturini aniqlash

5.2. Siqilgan (packed) fayllar belgilari

Zararli dastur mualliflari tahlilni qiyinlashtirish uchun faylni packer yoki kriptor vositalari bilan o‘raydi. Bunday faylda asl kod siqilgan yoki shifrlangan holatda bo‘ladi va faqat ishga tushganda xotirada tiklanadi. Shu sababli import jadvali ham odatiy ko‘rinishga ega bo‘lmaydi: unda ko‘pincha bir necha funksiya qoladi, ular orasida LoadLibraryA, GetProcAddress, VirtualAlloc, VirtualProtect va ExitProcess kabilar bo‘ladi. Bu funksiyalar asl importlarni ishga tushish vaqtida qo‘lda hal qilish uchun yetarli [3].

Shunday qilib, import jadvalining juda qisqaligi, bo‘limlarning bir xil bo‘lmagan nomlari va yuqori entropiya (ma’lumotlarning tasodifiy ko‘rinishi) kabi belgilar birgalikda faylning siqilgan bo‘lishi mumkinligini ko‘rsatadi. Bu faqat gumon, qat’iy xulosa emas: ba’zi qonuniy dasturlar ham siqilgan holda tarqatiladi.

5.3. Import hash (imphash)

Imphash — import jadvali asosida hisoblanadigan hash qiymati. U 2014-yilda Mandiant (FireEye) tadqiqotchilari tomonidan taklif qilingan va keyinchalik pefile kutubxonasiga qo‘shilgan. Hisoblash usuli: import qilingan har bir funksiya uchun DLL nomi (kengaytmasiz, kichik harflarda) va funksiya nomi kichik harflarda “dll.funksiya” ko‘rinishida yoziladi, hammasi vergul bilan birlashtirilib, MD5 hash olinadi. Muhim jihati shundaki, funksiyalar ketma-ketligi o‘zgarsa, hash ham o‘zgaradi [6].

Imphash bir oilaga mansub zararli dasturlarni guruhlashda foydali: bir dasturchi bir xil manba kodi va kompilyator bilan turli variantlar yaratganda, ularning fayl hashlari (MD5, SHA-256) har xil bo‘ladi, ammo import jadvallari o‘xshash bo‘lgani uchun imphash ko‘pincha bir xil chiqadi. Shu sababli VirusTotal kabi xizmatlar imphash bo‘yicha qidirishni qo‘llab-quvvatlaydi. Lekin imphash mukammal emas: packer qo‘llangan fayllarda hash packer imzosini ko‘rsatadi, zararli faylni emas, shuningdek oddiy dasturlar ham bir xil imphash ga ega bo‘lishi mumkin [6].

Imphashni qo‘llashda yana bir necha cheklov bor: (1) .NET Framework fayllarida import jadvali deyarli bir xil bo‘lgani uchun imphash ularni ajrata olmaydi (2.8-band); (2) bir dasturning 32 va 64 bitli yig‘malari turli imphash beradi (4.6-band); (3) import ro‘yxati yoki tartibidagi kichik o‘zgarish ham hashni butunlay o‘zgartiradi.

5.4. IAT va EAT hooking

Hooking — funksiya chaqiruvini o‘zgartirib, boshqaruvni begona kodga yo‘naltirish usuli. Import va eksport jadvallari bunga qulay nuqtalar hisoblanadi [10, 11].

IAT hooking. Dastur yuklangandan so‘ng IAT elementlarida haqiqiy funksiya manzillari turadi. Zararli kod (masalan, rootkit yoki brauzerdagi “man-in-the-browser” troyani) IAT dagi kerakli elementni o‘z funksiyasi manziliga almashtiradi. Natijada dastur, masalan, HttpSendRequestW ni chaqirganda aslida zararli kodga tushadi; u ma’lumotni o‘qib yoki o‘zgartirib, so‘ng asl funksiyani chaqirishi mumkin [10].

EAT hooking. Bunda DLL ning eksport manzillari jadvalidagi RVA qiymati almashtiriladi. Shundan so‘ng shu DLL dan funksiyani yangi qidirgan har qanday dastur zararli manzilga tushadi. Bu usulning cheklovi shundaki, u faqat hook o‘rnatilgandan keyin funksiya manzilini hal qilgan dasturlarga ta’sir qiladi [10].

Hookingni aniqlashda IAT dagi manzillar tegishli DLL xotira diapazoniga tushishi tekshiriladi. Agar element DLL ga emas, boshqa modulga yoki xotiraning shubhali joyiga ishora qilsa, bu hook alomati hisoblanadi.

Eksport jadvali bilan bog‘liq yana bir usul — DLL proxying (DLL side-loading bilan birga qo‘llanadi). Hujumchi qonuniy kutubxona nomi va eksport jadvaliga o‘xshash zararli DLL yaratadi: u asl kutubxonadagi barcha funksiyalarni eksport qiladi va ularni forwarded export orqali asl DLL ga yo‘naltiradi, shu bilan birga o‘z zararli kodini yuklash paytida ishga tushiradi. Dastur sezilarli o‘zgarishlarsiz ishlayveradi, shuning uchun bunday hujumni aniqlash qiyin. Tekshirish uchun kutubxonaning eksport jadvali asl nusxasi bilan solishtiriladi va ijobiy raqamli imzo mavjudligi tekshiriladi [12].

5.5. Importlarni yashirish usullari

Zararli dasturlar import jadvalidan o‘z maqsadini yashirish uchun bir necha usuldan foydalanadi [3, 12]:

– **Dinamik hal qilish.** Funksiyalar import jadvalida ko‘rsatilmaydi, ular ishga tushish vaqtida LoadLibrary va GetProcAddress yordamida topiladi. Natijada statik tahlilda faqat shu ikki funksiya ko‘rinadi [12].

– **API hashing.** Funksiya nomlari faylda ochiq yozilmaydi, ularning hash qiymatlari saqlanadi. Dastur yuklangan DLL larning eksport jadvallarini o‘zi aylanib chiqib, har bir nomning hashini hisoblaydi va mosini topadi. Shu bilan nomlar qatorlar ko‘rinishida (strings) topilmaydi [12].

– **PEB orqali modul qidirish.** Dastur yuklangan modullar ro‘yxatini jarayon muhiti bloki (PEB) orqali o‘qib, kutubxonalarning baza manzillarini LoadLibrary chaqirmasdan topadi, keyin eksport jadvalini o‘zi tahlil qiladi.

– **Ordinal bo‘yicha import.** Funksiya nomi o‘rniga ordinal ishlatilib, oddiy satr qidiruvi natija bermaydi.

Bu usullar import va eksport jadvallari tuzilishi haqidagi bilimning amaliy ahamiyatini ko‘rsatadi: zararli dasturning o‘zi ham aynan shu tuzilmalarni qo‘lda o‘qiydi, tahlilchi esa uning ishini tushunish uchun xuddi shu bilimga muhtoj.

5.6. Himoya va aniqlash choralari

Import va eksport jadvallari bilan bog‘liq xavflardan himoyalanish va ularni aniqlash uchun quyidagi choralar tavsiya etiladi:

– tahlil vositalarida (antivirus, sandbox, EDR) import jadvali, import hash va shubhali API kombinatsiyalarini avtomatik tekshirish;

– fayl entropiyasi, bo‘limlar nomlari va hajmlarini import jadvaliga qo‘shib baholash: bu qisqa import jadvali bilan birga siqilgan faylni aniqlashga yordam beradi;

– ishlayotgan jarayonlarda IAT va EAT yaxlitligini tekshirish: manzillar tegishli modul chegarasidan chiqib ketmasligi kerak;

– DLL qidiruv tartibini cheklash (SetDefaultDllDirectories va LOAD_LIBRARY_SEARCH_SYSTEM32 bayrog‘i) hamda imzolanmagan DLL larni yuklashni cheklovchi siyosatlardan (WDAC, AppLocker) foydalanish: bu DLL proxying va side-loading xavfini kamaytiradi; ASLR, DEP va CFG kabi umumiy himoyalar esa import va eksport jadvalini o‘zgartirishdan to‘g‘ridan-to‘g‘ri himoya qilmaydi, shuning uchun ular IAT/EAT yaxlitligini tekshirish bilan birga ishlatiladi;

– tahlil natijalarini bir manbaga tayanmasdan, bir nechta usul (statik, dinamik, repyutatsiya xizmatlari) bilan tasdiqlash.

5.7. Tahlil jarayonining tavsiya etilgan tartibi

Yuqoridagi bilimlarni amalda qo‘llashda shubhali PE faylni statik tahlil qilishning quyidagi tartibini tavsiya etish mumkin. Barcha harakatlar faylni ishga tushirmasdan, izolyatsiya qilingan (virtual) muhitda bajarilishi kerak:

– fayl turini aniqlash (PE32 yoki PE32+, EXE yoki DLL) va kompilyatsiya vaqtini (TimeDateStamp) ko‘rish;

– bo‘limlar nomlari, hajmlari va entropiyasini tekshirish; nostandart nomlar va juda yuqori entropiya siqilgan fayl alomati;

– import jadvalini chiqarib, DLL lar va funksiyalar ro‘yxatini 6-jadvaldagi guruhlar bilan solishtirish;

– agar importlar juda kam bo‘lsa, delay-load va dinamik hal qilish ehtimolini ko‘rib chiqish;

– DLL fayl bo‘lsa, eksport jadvalini o‘rganish: funksiya nomlari, forwarded exportlar, qonuniy kutubxonaga o‘xshashlik;

– imphash ni hisoblab, repyutatsiya xizmatlarida tekshirish va o‘xshash namunalarni qidirish;

– natijalarni dinamik tahlil (sandbox) ma’lumotlari bilan solishtirib, yakuniy xulosa chiqarish.

XULOSA

Ushbu ishda PE fayl formatidagi import va eksport jadvallarining tuzilishi batafsil tahlil qilindi. Ish davomida quyidagi asosiy xulosalarga kelindi.

Birinchidan, PE fayl sarlavhalar va bo‘limlardan iborat bo‘lib, jadvallarni topish uchun DataDirectory massivi va RVA ni offsetga aylantirish qoidasi asosiy vosita hisoblanadi. Ikkinchidan, import jadvali IMAGE_IMPORT_DESCRIPTOR massivi, ILT va IAT orqali tashkil etilgan: ILT funksiya nomlarini saqlaydi, IAT esa yuklash vaqtida haqiqiy manzillar bilan to‘ldiriladi. Bound import va delay-load importlar bu mexanizmning ishga tushirish tezligi va resurs sarfini optimallashtirishga qaratilgan variantlaridir.

Uchinchidan, eksport jadvali uchta massiv — EAT, nomlar massivi va ordinallar massivi — yordamida funksiyalarni nom yoki ordinal bo‘yicha topish imkonini beradi, forwarded export esa funksiyani boshqa kutubxonaga yo‘naltiradi. To‘rtinchidan, bu jadvallar zararli dasturlarni statik tahlil qilishda dastur imkoniyatlari, siqilgan fayl belgilari va oilaviy o‘xshashlik (imphash) haqida qimmatli ma’lumot beradi, shu bilan birga IAT/EAT hooking, dinamik hal qilish va API hashing kabi usullar orqali hujumchilar tomonidan suiiste’mol ham qilinadi.

Beshinchidan, to‘rtta haqiqiy PE fayl (EXE va DLL) ustida o‘tkazilgan tajriba nazariy bayonni tasdiqladi: qo‘lda hisoblangan offsetlar, deskriptorlar va import sonlari pefile natijasi bilan mos keldi; Microsoft linkeri import jadvalini .rdata ichiga joylashtirishi va zamonaviy fayllarda API Set nomlari ustunligi aniqlandi; yozilgan my_imphash funksiyasi get_imphash() bilan to‘liq mos keldi. Oltinchidan, zararsiz fayllarda ham IsDebuggerPresent va OpenProcess kabi “shubhali” funksiyalar uchragani import asosidagi belgilarni yakka holda emas, majmua sifatida baholash zarurligini ko‘rsatdi.

Ishning cheklovlari. Tajriba faqat to‘rtta zararsiz faylda o‘tkazildi (zararli namunalar ishlatilmadi), shuning uchun aniqlash samaradorligi statistik baholanmadi; ordinal bo‘yicha import qiluvchi, bound yoki delay-load importli fayllar tajribaga kiritilmadi; .NET fayllari alohida tahlil qilinmadi.

Shunday qilib, import va eksport jadvallari tuzilishini chuqur bilish zararli dasturlar tahlili, antivirus texnologiyalari va raqamli kriminalistika sohalarida ishlaydigan mutaxassis uchun zarur asos hisoblanadi. Keyingi tadqiqotlarda bu jadvallardan olingan belgilar asosida zararli dasturlarni mashinaviy o‘rganish usullari bilan oilalarga ajratish yo‘nalishini ham rivojlantirish mumkin [13–15].

FOYDALANILGAN ADABIYOTLAR

1. Microsoft Corporation. PE Format. Microsoft Learn, Windows Win32 hujjatlari (learn.microsoft.com/en-us/windows/win32/debug/pe-format).
2. Pietrek M. An In-Depth Look into the Win32 Portable Executable File Format. MSDN Magazine, 2002.
3. Sikorski M., Honig A. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. — San Francisco: No Starch Press, 2012.
4. Eilam E. Reversing: Secrets of Reverse Engineering. — Indianapolis: Wiley Publishing, 2005.
5. Yosifovich P., Ionescu A., Russinovich M., Solomon D. Windows Internals, 7th Edition. — Microsoft Press, 2017.
6. Mandiant (FireEye). Tracking Malware with Import Hashing. Mandiant blogi, 2014 (mandiant.com/resources/blog/tracking-malware-import-hashing).
7. Carrera E. pefile — Python PE parsing module. Loyiha hujjatlari (github.com/erocarrera/pefile).
8. Microsoft Corporation. Linker support for delay-loaded DLLs. Microsoft Learn (learn.microsoft.com/en-us/cpp/build/reference/linker-support-for-delay-loaded-dlls).
9. Microsoft Corporation. Windows API sets. Microsoft Learn (learn.microsoft.com/en-us/windows/win32/apiindex/windows-apisets).
10. Ligh M.H., Case A., Levy J., Walters A. The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. — Indianapolis: Wiley, 2014.
11. Hoglund G., Butler J. Rootkits: Subverting the Windows Kernel. — Upper Saddle River: Addison-Wesley, 2005.
12. MITRE ATT&CK. Techniques T1055 (Process Injection), T1574.002 (DLL Side-Loading), T1027.007 (Dynamic API Resolution) (attack.mitre.org/techniques/).
13. Anderson H.S., Roth P. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. — arXiv:1804.04637, 2018.
14. Raff E., Barker J., Sylvester J., Brandon R., Catanzaro B., Nicholas C. Malware Detection by Eating a Whole EXE. — AAAI-2018 Workshop on Artificial Intelligence for Cyber Security, 2018.
15. Saxe J., Berlin K. Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. — 10th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 2015, pp. 11–20.
16. Saxe J., Sanders H. Malware Data Science: Attack Detection and Attribution. — San Francisco: No Starch Press, 2018.
17. TIS Committee. Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, Version 1.2. — 1995.