Legatura dinamica(comparatie linux vs. windows)

Aceasta tema dezbate conceptul librariilor share-uite in cele 2 sisteme de operare cel mai des intalnite Windows si Linux, si ofera o incursiune printre structurile de date variate pentru a explica cum legarea dinamica este facuta in aceste sisteme de operare. Lucrarea va fi folositoare pentru programatorii interesati in implicatiile legate de securitate si viteza relativa a legaturii dinamice, si presupune niste cunostinte anterioara a acestui subiect.

In cele ce urmeaza vom discuta despre cum functioneaza in wondows iar mai apoi vom continua sa comparam cele 2 medii.

Structurile de date cu fisiere executabile portabile Windows (PE)

Stim ca o sectiune este o bucata de cod sau informatie legata intr-un mod logic, si mai stim ca informatia unor tabele de importuri a unui executabil sunt intr-o sectiune. In acest referat ne uitam la niste sectiuni in fisierele Windows PE.

Sectiunea de exporturi(.edata)

Sectiunea .edata incepe cu structura de sirectoare export IMAGE_EXPORT_DIRECTORY. Directorul de exporturi contine RVA-urile(adresele virtuale relative) ale Tabelei de Adrese de Export. Acesta contine adrese ale punctelor de intrare exportate, informatie exportata si valori absolute. Un numar ordinal este folosit pentru indexarea tabelei de adrese. BAZA ORDINALA trebuie sa fie extrasa din numarul ordinal inainte de indexarea inauntrul tabelei.

Pointeri tabele de nume export: Acest vector contine adrese in tabele de nume export. Pointerii sunt legati la baza imaginii si sunt ordonati lexical pentru a inlesni cautarea binara. Tabela de nume export contine nume(ASCII) pentru intrarie exportate in imagine.

Tabela ordinala de export: Pointerii tabelei de nume de export si tabela de export ordinala din cei 2 vectori paraleli. Vectorul tabela ordinala de export contine numarul ordinal asociat cu numele exportat referit de pointerii tabela de nume de export. Numarul ordinal va servi ca index in EAT.

Sectiunea de importuri(.idata)

Sectiunea .idata face conversia a ceea ce face sectiunea .edata, descrisa mai sus. Ea mapeaza simboluri/numere ordinale inapoi in RVA-uri. Sectiunea .idata incepe cu o tabela de directoare de importuri IMAGE_IMPORT_DIRECTORY . Tabela de directoare de importuri este compusa dintr-un vector de structuri IMAGE_IMPORT_DESCRIPTOR, cate unul pentru fiecare executabil importat. IMAGE_IMPORT_DESCRIPTOR contine RVA-uri de tipul:

Tabela de referinte de import: Aceasta este un vector de structuri IMAGE_THUNK_DATA. Structura contine numerul ordinal sau numele RVA pentru fiecare functie importata. Tabela identifica simbolurile de importat, cu intrarile in tabela import de referinta fiind paralel cu acelea din tabela de adrese import(IAT). Daca bitul cel mai semnificativ este setat atunci ceilalti biti reprezinta sunt numarul ordinal. Altfel intrarea este RVA-ul unei intrari in tabela de referinta pentru nume .

Tabela de adrese de import: Aceasta este tot un vectori de structuri IMAGE_THUNK_DATA. Initial amandoua tabela de referinta(Lookup table) si tabela de adrese import(IAT) contin intrari simileare. Loader-ul introduce adresele fiecarei rutine importate in aceasta tabela, in timp ce intrarile din tabela de referinta a intrarilor(Import Lookup Table) retine informatia originala cum era initial. Vom vedea de ce linker-ul mentine informatia originala mai tarziu cand vom discuta despre operatia numita binding.

Tabela de nume indiciu(Hint-Name Table): Tabela contine un indiciu de 4 byte-uri urmat de un simbol nume nul. Valuarea indiciului este folosit pentru indexarea vectorilor de pointeri catre Tabela de nume de export(Export Name Table) permitand importuri mai rapide. Indiciul va fi corect daca DLL-ul nu s-a schimbat sau cel putin lista sa de simboluri exportate nu s-a schimbat. Daca indiciul este incorect atunci cautarea binara is facuta in Tabela de pointeri nume(Export Name Pointer table).

Cum functioneaza lucrurile

Incarcand un executabil de Windows si un DLL este simimlar cu incarcarea unui program ELF legat dinamic in Liunx. Diferenta este ca link-erul este o parte a kernel-ului insusi. Prima data

kernel-ul mapeaza in executabilul indrumat de header-ele PE. Loader-ul se uita la IAT al modulului si determina daca DLL-ul depinde de DLL-uri aditionale, si daca loader-ul le mapeaza si pe acestea. Acest proces continua pana cand toate modulele dependente au fost mapate in meorie.

O functie importata poate fi listata dupa nume sau dupa numarul ordinal. Numarul ordinal reprezinta pozitia ei in Tabela de adrese exportate al DLL-ului. Daca este listata de nume, loader-ul face o cautare binara a Tabelei de pointeri nume exportate a DLL-ului corespunzator pentru a cauta index-ul la care simbolul este gasit.Dupa aceea el foloseste acel index ca un index in Tabela de numere ordinale exportate pentru a primi numarul ordinal care este folosit ca un index in Tabela de adrese exportate. Adaugand RVA-ul simbolului gasit din Tabela de Adrese de export (EAT) la adresa de incarcare a DLL-ului va rezulta adresa absoluta pe care loader-ul o scrie la intrarea corespunzatoare in tabela de adrese de import (IAT)

Incarcarea lenesa in Windows

Un DLL incarcat cu delay are o structura ImgDelayDescr similara cu structura de directoare de date de import .idata dar el nu este in sectiunea .idata. ImgDelayDescr contine adresele unei tabele de adrese de import(IAT) si a unei tabele de nume de import(INT) pentru DLL. Aceste tabele sunt identice in format cu cele normale de importuri, dar mai degraba ele sunt scrise si citite de codul librariilor runtime decat sistemul de operare. Cand apelezi un API pentru prima oara dintr-un DLL incarcat cu delay, libraria runtime incarca DLL(daca este necesar), primeste adresa, si depoziteaza tabela de adrese de import incarcata cu delay astefl incat apelarile viitoare merg direct la fisierul API.