Practical Malware Analysis & Triage preparation

Course introduction

Ammar Jokhadar

Reverse Engineer, Exploit developer
Security researcher & Online instructor.

معلومات بسيطة عن الملفات التنفيذية exe

قبل نبدأ في اي شي ضروري نفهم بعض الاساسيات, ملفات الـexe اصلا عبارة عن ايش؟ وكيف تشتغل وعشان نفهم هذا الشي جهزت شوية سلايدات نتكلم فيها بشكل بسيط وواضح عن ايش هي هذي الملفات وكيف الكمبيوتر يفهمها ويشغلها.

كود بسيط C++

الكود البسيط الي امامنا الان بسيط ومهمته واضحه جداً, يطبع لنا رساله بسيطة ثم يخرج
لكن هل فعلا الكمبيوتر يقدر يفهم هذا الكلام؟

```cpp
#include <cstdio>

int main() {
    puts("Hello world I'm a PE file :)");
    return 0;
}
```

Compiler

بالتأكيد الحاسب غير قادر على فهم الكود الي كتبناه قبل شويه, طيب دام الكمبيوتر غير قادر على فهم الكود كيف احنا نسوي له كومبايل ويتنفذ ويطبع لنا الرسالة بشكل صحيح؟

هنا ببساطة لازم نعرف ان الحاسب فقط يفهم رمزين الصفر والواحد (0,1) وهذا المفهوم الي اغلبنا نعرفه مسبقاً, يسمى هذا النظام نظام العد الثنائي (Binary) ومن هنا نعرف ان الكود الي نكتبه مصيره يتحول الى واحدات واصفار بطريقه وترتيب معين يفهمها الحاسب فقط.
لذلك الكومبايلر (Compiler) يقوم بعملية تحويل مانكتبه من سطور برمجية إلى Assembly لكن وش هي Assembly ؟

Assembly

لو قد مرت عليك اسيمبلي بتعرف على طول ان Assembly ماهي عبارة عن واحدات واصفار, وانها تحتوي على مجموعه كبيرة من التعليمات المكتوبة امثلة (mov,pop,add) إذا ليه الكومبايلر يحول الكود الى assembly واحنا متفقين ان الحاسب فقط يقدر يفهم واحدات واصفار؟

لو فكرنا شويه وتخيلنا قد ايش بيكون صعب نكتب كود كامل عباره عن واحدات واصفار بنعرف اننا نحتاج حل واسيمبلي هي كانت الحل بحيث ان اللغه القريبه جدا من الباينري تسمح لنا نكتب الكود ك نصوص وهي تهتم بتحويله الى واحدات واصفار على حسب معمارية المعالج, لان تعليمة بسيطة في لغة اسيمبلي مثل : mov eax, 0x1337 على سبيل المثال
بينتج عنها :
10111000001101110001001100000000000000
ومن هنا نقدر نفهم قد ايش صعب تكتب كود كامل في الـBinary system

معلومات بسيطة عن اسيمبلي

```
; Attributes: bp-based frame fuzzy-sp

; int main()
public _main
_main proc near
; __unwind {
push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 10h
call    ___main
mov     dword ptr [esp], offset Buffer ; "Hello world I'm a PE file :)"
call    _puts
mov     eax, 0
leave
retn
; } // starts at 401410
_main endp
```

نقدر نشوف الان ايش صار في الكود بعد عملية الـCompiling ونشوف امامنا كود Assembly
الكود الفعلي عبارة عن Binary لكن في هذه الصورة قمت باستعمال IDA وهي اداة للهندسة العكسية تقوم بعكس عملية الـAssembler
حيث ان الاسمبلر يحول اسيمبلي كود الى باينري
اما IDA تقوم باخذ الباينري كود وعرضه ك اسيمبلي كود عشان نقدر نفهمه ونحلله

عملية تحويل الكود من اسيمبلي الى Binary code تسمى Compiling ماتقوم به IDA يسمى Decompiling

ليش نحتاج الـdecompilers ؟

ذكرنا مسبقاً ان الكود بالكامل بيتحول الى باينري او بمسمى اخر ماشين كود (Machine code), بالتالي لو حاولنا نفهم بشكل مباشر بيكون اشبه بالجنون لانه بيكون غير قابل للقراءة من البشر, وهنا يجينا الـDecompiler ويلعب اهم دور حيث انه ياخذ هذي البايتات الي انتجها الاسمبلر (Assembler) ويرجعها لنا على شكل Assembly code بالتالي نقدر نفهم الكود ونحلله بشكل طبيعي.