Introduction To Disassembly

Author. Ahmad AlFareed
Section Reverse Engineering - Tools
rETKit

1. Disassembly Theory

من درس البرمجة او البرمجة المتقدمة يعلم ما يوجد هنا من مصطلحات لكن سنقوم بشرحها للذين تجاهلوها او نسوها.

1.1 First-generation languages

هذه هي ادنى اشكال اللغة وتكون عموما على هيئة اصفار و واحاد [0,1] (Binary) او من بعض الاشكال المختصرة مثل السداسي العشري (Hex)و لا يمكن قراءتها الا بواسطة برامج مخصصة لذالك. الامر في هذا المستوى غالبا ما تكون مربكة لانه غالبا ما يكون صعب التمييز بين البيانات والتعليمات لانه كلشئ يبدو متماثلا الى حد كبير. ويمكن ايضا الاشارة الى هذا المستوى بـ machine languages ويشار الى هاذي البرامج بـ binaries .

1.2 Second-generation languages

تعرف بـ assembly languages او بـ second-generation languages وهي عبارة بحث عن في جدول بعيدا عن الـ machine language وتقوم بتعيين بشكل عام bit patterns او operation codes (opcodes) الى character sequences تسمى mnemonics .

1.3 Third-generation languages

تتخذ هاذي اللغات قدرة تعبيرية تكون اكثر تقريبيه للغات الطبيعية البشرية الي بنستخدمها بمعنى ادق هي اللغات الـ High-Level مثل C/++ , C# & Python . وتستخدم Compilers عشان تترجمها الى لغات التجميع ثم الى لغة الالة ثم التشغيل.

1.4 Fourth-generation languages

وفي اكثر من هيك بكثير مثل المستوى الرابع والخامس لكن الكتاب لا يغطيهم. مثال سريع في هذا المستوى مثل اللغات الي مصممة لتطبيقات محددة مثل SQL .

2.The What of Disassembly

في مفهوم تطوير البرمجيات يتم استخدام الـ Compilers,Assemblers & Linkers كل وحده لحالهم او كلهم متجمعين في برنامج واحد لانشاء برنامج قابل للتنفيذ بناء على نظام التشغيل.

2.1 The compilation process is lossy

على مستوى الـ machine language level لا توجد اسماء متغيرات او وظائف ويمكن تحديد نوع واحد من المتغيرات من خلال فهم كيفية استخدام هذا النوع للبيانات بدلا من ذالك ما يستخدم نوع صريح (type declarations) في الـ Machine Language عندما تلاحظ Data كـ 32-bit ستحتاج الى القيام باعمال الاستقصائية (investigative) بمعنى تتبع دقيق لاكتشاف ان كانت تستخدم على سبيل المثال فاصلة عائمة (floating-point) او مؤشر (Pointer) او عدد صحيح(Integer) .

2.2 Compilation is a many-to-many operation.

هذا يعني انه يمكن ترجمة (source program) الى لغة التجميع بعدة طرق مختلفة ويمكن ترجمة لغة الالة مرة اخرى الى source بعدة طرق. ونتيجة لذالك فمن الشائع جدا ان تجميع ملف (compiling a file) وفك ترجمته على الفور قد يؤدي الى ملف مصدر مختلف تماما عن الملف الذي تم ادخاله.

2.3 Decompilers are very language and library dependent

بنسبة الي الـ Decompilers لحد الان ضعيف جدا لكن مفيدين لؤلاك الهاوين في الهندسة العكسية لعدة اسباب ابرزها من الممكن تتم كتابة برنامج بلغة #C وتقوم بعمل Decompiling عن طريق IDA سيخرج لك الاغرب على الاطلاق. (بناء على انه يستخدم C Decompiler). وايضا عيب اخر ممكن ان يخطأ باستنتاج العديد من المواضيع منها الـ Pointer الموجوده في البرنامج. فقط الفائدة الواضحه لها هو توضيح الـ Parameters & Windows API's الموجودة.

3.The Why of Disassembly

الـ disassembly tools الغرض منها هو تسهيل فهم وفهم البرنامج عندما لا يكون هناك source code . تشمل المواقف الشائعه ليش نستخدمها مثال :

3.1 Analysis of malware (Malware Analysis)

اذا ما كنت تتعامل مع script-based malware مثل كان البرنامج الضار على هيئة ثنائي يكون هنا افتقار لالكود المصدري للبرنامج الضار فانك ستواجه مجموعة محدودة جدا من الخيارات لكيفية تصرف البرنامج الضار بزبط. مثل اذا كنت تتعامل مع Ransomware المجموعات المحدودة التي يمكنك معرفتها انه اولا برنامج ضار يحتوي على برمجيات خبيثة ويحتاج الى مفتاح ويوجد به نصوص تهدد ومؤقت عند انتهائه يزداد المبلغ الفدية وطرق الدفع لكن لا تعلم ما اذا كان هذا المفتاح كيف ينتج عن طريق الـ Client-Side او Server-Side ما هي الخوارزميات المستخدمة والخ.
الطريقيتين الرئيستين لتحليل البرنامج الضار هما dynamic analysis و static analysis تم شرحها جميعها هنا (rETKit Malware Analysis) الـ Dynamic analysis يتضمن السماح للبرنامج الضار بالتنفيذ في بيئة افتراضية لتجنب الاعراض الخبيثة في البيئة الحقيقة ويتم التحكم بها في عناية مع تسجيل جميع التحليلات وملاحظة سلوكها .
الـ static analysis يحاول فهم سلوك البرنامج ببساطة عن طريق قراءة رمز البرنامج من دون تشغيله .

3.2 Analysis of closed-source software for vulnerabilities (Vulnerability Analysis)

من اجل التبسيط دعونا نقسم عملية التدقيق الامني (security-auditing) باكملها الى ثلاثة خطوات : اكتشاف الثغرات (vulnerability discovery) و تحليل الثغرات الامنية (vulnerability analysis) و (exploit development). تنطبق نفس العمليات سواء كان البرنامج بمصدر او من غير لكن يزداد الصعوبة عندما يكون ثنائي لوحده فقط من غير كود مصدر. الخطوة الاولى هي اكتشاف حالة يمكن استغلالها في البرنامج. يتم التحقيق من تقنيات شائعة باستخدام dynamic techniques مثل fuzzing ويمكنك ايضا عن طريق static analysis . بمجرد اكتشاف مشكلة غالبا ما تكون هناك حاجة الى مزيد من التحليل لتحديد ما اذا كانت المشكلة قابلة للاستغلال وتحت اي ظروف.
توفر الـ Disassembly مستوى التفصيل المطلوب لفهم كيفية اختيار المترجم لتخصيص المتغيرات في البرنامج مثل قد

يكون من المفيد معرفة ان المصفوفة حجمها byte character array-70 وتوفر ايضا تحديد لترتيب المتغيرات المعلنة global or local . بالمختصر فقط باستخدام disassembler او debugger يمكن بناء برنامج استغلال.

3.3 Analysis of closed-source software for interoperability (Software Interoperability)

عندما يتم اصدار برنامج بهيئة ثنائي يصعب جدا على المنافسين انشاء برنامج التفاعل معه او توفير اضافة لذالك البرنامج. من الامثله الشائعة الـ driver code الذي تم اصداره للـ hardware المدعومة على نظام اساسي واحد فقط. عندما يكون البائع بطيئا في الدعم. يرفض دعم الـ Driver في منصات اخرى فقد يكون هناك جهد كبير في الهندسة العكسية من اجل تطوير هذا لدعم hardware . في هذه الحالة يعد تحليل التعليمات البرمجية الثابتة (static code analysis) هو الحل الامثل وغالبا ما يجب يتجاوز الـ software driver ويدخل الى لفهم embedded firmware .

3.4 Analysis of compiler-generated code to validate compiler performance/ correctness (Compiler Validation)

نظرا لان الغرض من المترجم او الـ assembler هو انشاء الـ machine language الـ disassembly tools بحاجة لها للتحقق من المترجم يعمل ضمن ضوابط التصميم وبهدف التحسين ومن الوجهات النظر الامنية لانه من الممكن ان يولد الـ Compiler ثغره عند تجميع البرنامج ويؤدي الى ادخال برمجيات يمكن استغلالها. Compiler Bugs

3.5 Display of program instructions while debugging (Debugging Displays)

استخدام الـ Debugging بجانب الـ disassemblers امر مهم لعدة اسباب منها لتحديد الوظيفة بشكل اسرع واسهل وفهمها وفهم ما يوجد من معاملات في الوظيفة ومعرفة حدود الوظيفة .

4.The How of Disassembly

الان بعد ان اصبحت على دراية ببعض المواضيع المهمة حان الوقت الان عشان نبدأ نتعامل مع disassembly وكيف العملية بالفعل تتم. مهمة الـ disassemblers مهمة شاقة على سبيل المثال ياخذ ملف 100 كيلو بايت وقم بتمييز الكود عن البيانات وقم بتحويلها الى كود تجميع لعرضة ويرجى عدم تفويت شيء. وتحديد موقع الوظائف والتعرف على recognize jump tables وتحديد المتغيرات المحلية الخ. جودة الـ disassemblers تكمن في خوارزيماتا . سنناقش في هذا القسم اثنين من الخوارزميات الاساسية المستخدمة في disassembling machine code .

4.1 A Basic Disassembly Algorithm

للمبتدئين بدنا نوخذ الخوارزمية الاساسية الي هو ياخذ input الـ machine language ويخرجه assembly language كـ output . قبل كلشئ في خطوات اساسية لازم نعرفها :

Step 1 :

الخطوة الاولى في عملية الـ disassembly هي تحديد منطقة التعليمات البرمجية المراد تفكيكها . عادة ما يتم خلط الـ Data بـ الـ Instructions ومن المهم التميز بين الاثنين. في الحالة الاكثر شيوعا عند عمل disassembly لـ executable سيتوافق الملف مع تنسيقات مثل Protable Executable PE المستخدمة في Windows . تحتوي هاذي التنسيقات على mechanisms او بتكون على شكل file headers . لتحديد موقع الاقسام التي تحتوي على التعليمات البرمجية و الـ Entry Point وما الى ذالك.

Step 2 :

الخطوة التالية هي قراءة القيمة الموجودة في هذا العنوان او (file offset) واجراء بحث في جدول Table يكون مطابق لـ opcodes مع الـ Assembly Instructions .

Step 3 :

بمجرد فك التعليمات و اي المعاملات المطلوبة يتم عمل formatted and output في ما يعادل ما تم فكه ويظهره كـ جزء من فك التشفير يمكنك اختيار اي assembly language output syntax سواء كان Intel او AT&T .

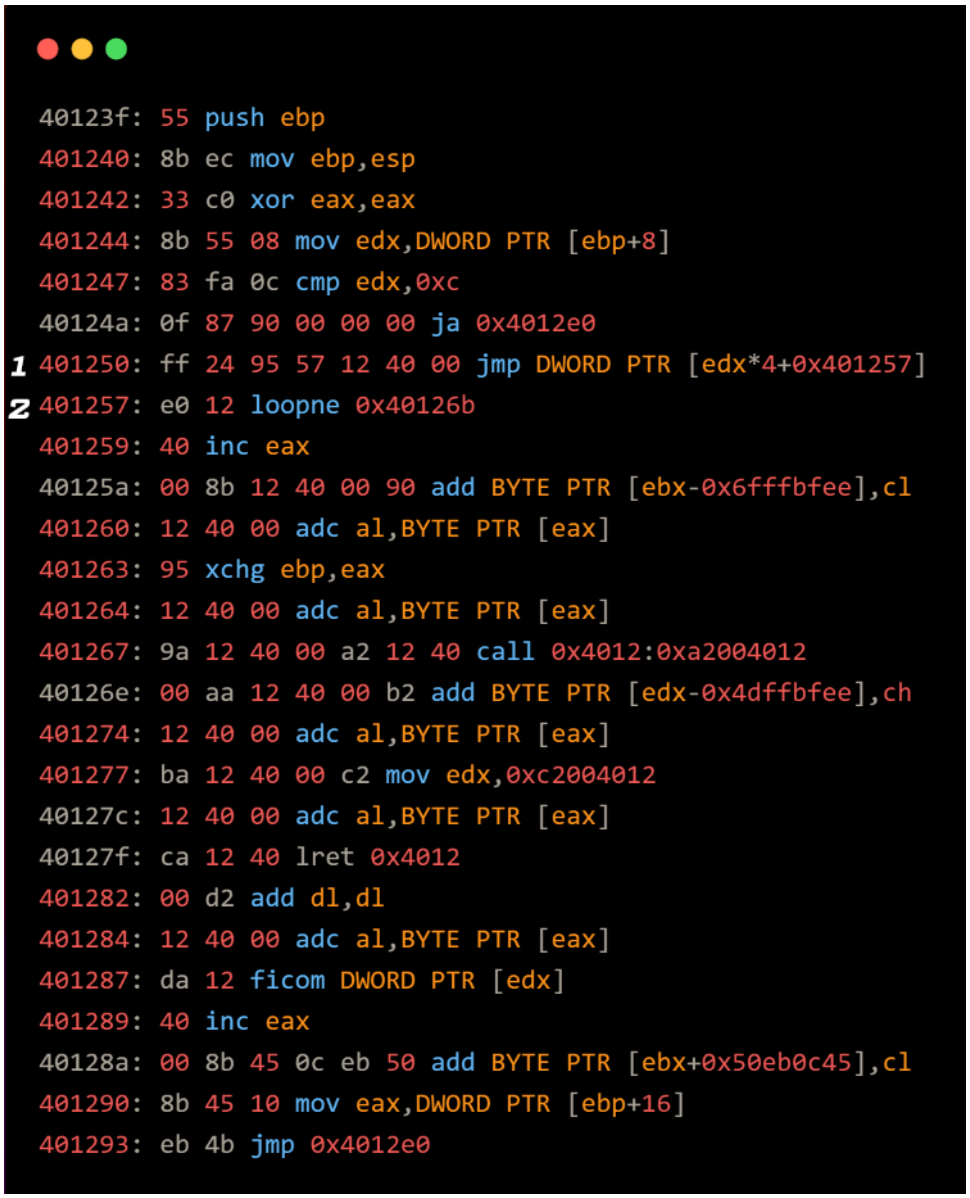
Step 4 :

بعد اخراج التعليمات (output of an instruction) نحتاج الى الانتقال الى التعليمات التالية وتكرار نفس العملية السابقة حتى نقوم بتفكيك كل تعليمات الملف. توجد خوارزميات مختلفة لتحديد مكان بدء التفكيك وكيفية اختيار التعليمة التالية التي سيتم تفكيكها وكيفية التميز بين البيانات والتعليمات وكيفية تحديد متى تم تفكيك التعليمة الاخيرة. في خوارزميتان هم linear sweep و recursive descent .

Linear Sweep Disassembly

الـ linear sweep disassembly algorithm تتبع لتحديد موقع التعليمات المراد تفكيكها حيث تنتهي من تفكيك احدى التعليمات وتبدأ بالاخرى. ونتيجة لذالك القرار الاكثر صعوبة هي اين تبدأ . الحل المعتاد هو افتراض ان كل ما هو موجود في الـ sections يكون marked كـ code يتم تحديدة عادة بواسطة الـ files headers وهو يمثل الـ machine language instructions . يبدأ بتفكيك بالبايت الاول في code section ويتحرك بطريقة خطية عبر القسم هذا ويفكك التعليمات واحده تلو الاخرى حتى الوصول الى النهاية. هي تعتبر غاشمة ولا يوجد اي خوارزميات اخرى يمكنها فهم تدفق البرنامج مثل من خلال التعرف التعليمات الغير خطية مثل الـ branches . اثناء عملية التفكيك يمكن الاحتفاظ بؤشر لتحديد بداية التعليمات التي يتم تفكيكها حاليا. كجزء من عملية التفكيك يتم حساب طول كل تعليمة واستخدامها لتحديد موقع التعليمة التالية التي سيتم تفكيكها. هو الميزة الرئيسية فيها هي توفر تغطية كاملة لتفكيك القسم التعليمات البرمجية في البرنامج. ويوجد بها العديد من العيوب اولها طريقة المسح الخطي هو عدم مراعاة ان البيانات قد تكون مختلطة بالكود.

1-1: Linear sweep disassembly



```
40123f: 55 push ebp
401240: 8b ec mov ebp,esp
401242: 33 c0 xor eax,eax
401244: 8b 55 08 mov edx,DWORD PTR [ebp+8]
401247: 83 fa 0c cmp edx,0xc
40124a: 0f 87 90 00 00 00 ja 0x4012e0
1 401250: ff 24 95 57 12 40 00 jmp DWORD PTR [edx*4+0x401257]
2 401257: e0 12 loopne 0x40126b
401259: 40 inc eax
40125a: 00 8b 12 40 00 90 add BYTE PTR [ebx-0x6fffbfee],cl
401260: 12 40 00 adc al,BYTE PTR [eax]
401263: 95 xchg ebp,eax
401264: 12 40 00 adc al,BYTE PTR [eax]
401267: 9a 12 40 00 a2 12 40 call 0x4012:0xa2004012
40126e: 00 aa 12 40 00 b2 add BYTE PTR [edx-0x4dffbfee],ch
401274: 12 40 00 adc al,BYTE PTR [eax]
401277: ba 12 40 00 c2 mov edx,0xc2004012
40127c: 12 40 00 adc al,BYTE PTR [eax]
40127f: ca 12 40 lret 0x4012
401282: 00 d2 add dl,dl
401284: 12 40 00 adc al,BYTE PTR [eax]
401287: da 12 ficom DWORD PTR [edx]
401289: 40 inc eax
40128a: 00 8b 45 0c eb 50 add BYTE PTR [ebx+0x50eb0c45],cl
401290: 8b 45 10 mov eax,DWORD PTR [ebp+16]
401293: eb 4b jmp 0x4012e0
```

الكود هذا توضح مخرجات دالة مفككة باستخدام الـ linear sweep disassembler . الوظيفة هاذي تستخدم switch statement والـ Compiler قد استخدم الـ jump table في حالة تنفيذ الـ switch case . الـ jmp statement في [1] عند عنوان 401250 . تشير address table في [2] عند عنوان 401257 . وايضا الـ disassembler يعامل Y كما لو كان تعليمة ويقوم بشكل غير صحيح بانشاء تمثيل بلغة Assembly . الـ Linear sweep يستخدم gdb و WinDbg و ايضا objdump لكن هي افضل.

Recursive Descent Disassembly

الـ Recursive descent ياخذ اسلوبا مختلفا لتحديد موقع التعليمات البرمجية. الـ Recursive descent يركز على مفهوم تدفق التحكم (control flow) الذي يحدد ما اذا كان يجب تفكيك التعليمات ام لا بناء على ما اذا كان يتم الرجوع

اليها بواسطة تعليمات اخرى. لفهم الـ recursive descent من المفيد تصنيف التعليمات وفقا لكيفية تاثيرها على CPU instruction pointer .

Sequential Flow Instructions

الـ Sequential flow instructions تقوم بتمرير التنفيذ الى التعليمات التي يتبعها. امثلة الـ sequential flow instructions تتضمن التعليمات الحسابية (arithmetic instructions) مثل الـ add . تعليمات الـ register-to-memory transfer مثل mov وتعليمات الـ stack-manipulation operations مثل الـ push & pop etc. . بالنسبة لمثل هاذي التعليمات يستمر التفكيك كما هو الحال مع الـ linear sweep .

Conditional Branching Instructions

الـ Conditional branching instructions مثل الـ x86 jnz مسارين محتملين للتنفيذ.

75 cb JNZ rel8 D Valid Valid Jump short if not zero (ZF=0).

اذا تم تقيم هذا الشرط على انه صحيح فسيتم اخذ الفرع المناسب ويجب تغيير الـ instruction pointer ليعمل عملية العكس على هذا الفرع المستهدف وايضا يشير عليه. ومع ذالك اذا كان خطأ يستمر في التنفيذ بطريقة linear ويمكن استخدام منهجية linear sweep لتفكيك التعليمة التالية. ولان في static analysis من الصعب تحديد بالظبط ان كان على صواب او خطأ فان الخوارزمية تفكك كلا المسارين.

Unconditional Branching Instructions

الـ Unconditional branches لا تتبع نموذج الـ linear flow model وبالتالي يتم التعامل معها بشكل مختلف من خلال خوارزمية recursive descent . كما هو الحال مع sequential flow instructions يمكن ان يتدفق التنفيذ الى تعليمة واحدة فقط ومع ذالك لا يلزم ان تتبع هذه التعليمات تعليمات الـ branch instruction مباشرة.

Twitter : https://twitter.com/dr_retkit
YouTube : https://www.youtube.com/@retkit1823