Статья Полиморфный генератор мусора

Simple_trash_gen_module - Модуль для полиморфной генерации "мусорного кода".

Что такое "мусорный код" и зачем он нужен ?

Мусорный код - Это имитация работы программы, т.е. это исполнение различных ассемблерных команд в случайном порядке (Желательно генерировать их в момент работы программы, как это сделать ниже), также можно вызывать различные Windows API функции, в случайном порядке в момент работы программы.

Зачем это нужно ?

1)Это может использоваться при создании протекторов, как известно, что-бы исследовать программу реверсер открывает программу в отладчике в момент исполнения, видя там дизасемблированный код, у реверсера уже складывается представление о там-что делает программа.

Теперь представьте, что в момент исполнения программы, наша программа начнёт имитировать действия, в случайном порядке, более-того, при каждом запуске действия будут разные...)

2)Также это может использоваться для "темных дел", в частности для обхода детекта антивирусов, ведь антивирус по простому, это тот-же "реверсер", который дизассемблирует программу, смотрит-что там и по разным критериям выдает детект (Сигнатура, поведение программы и т.д.).

А теперь представьте, что мы нагенерируем 1000-чи безобидных инструкций и будем вызывать 100-200 API и всё это в случайном порядке и при запуске программы, каждый-раз программа будет заниматься ерундой...)

Понятно, что генерировать такой код нужно в автоматическом режиме, не будете-же вы вручную размещать 1000-чи инструкций, если апи ещё можно повызывать из программы, то с инструкциями сложнее...

В общем-то для этого и написан такой модуль.)

Описание модуля:

1)Генерация инструкций:

Генератор инструкций лучше писать на ассемблере. Почему ассемблер ? Потому-что если вы хотите сделать это например на Си, то вам придется дергать кучу Windows API, таких-как VirtualProtect, что уже не делает нашу программу так-сказать "чистой", у антивирусов и у реверсера уже будет подозрения зачем это делается.)

Поэтому как-раз для этой задачи ассемблер идеальный вариант.

В исходниках за это отвечает файл "fake_instructions.asm", на самом деле код не сложный там коментарии, единственное расскажу по процедуре сборки.

Для компиляции ассемблерного кода я использую компилятор Fasm (В исходниках папка fasmw17309), был выбран именно этот компилятор, т.к. он во первых "легковесный", а во вторых имеет различные макросы, которые автоматизируют многие задачи, короче с ним проще разобраться, кто плохо знает ассемблер, но в то-же время можно писать различные сложные программы.

Т.к. я использую среду Visual C++ и предполагается, что основная программа будет на C++, то ассемблерный код собирается в библиотеку .obj, используется формат MS COFF (Про этот формат можно почитать в сети).

Далее эта библиотека линкуется стандартным линкером Visual C++, достаточно подключить её к проекту (в исходниках уже сделаны эти настройки).

Кстати, если вы любитель других языков например Делфи, C# прочее...

То вы можете использовать эту библиотеку, единственное нужно учитывать "Соглашение о вызовах функций"(Про это можно почитать в сети), т.е. достаточно изменить прототипы ассемблерных функций.

Описание интерфейсных функций, те-которые можно вызывать из программы на высокоуревновом языке:

```
;-------------------------------------------
; Интерфейсная функция, для получения случайного числа в нужном интервале ;
; stdcall do_Random_EAX,min,max ; на выходе EAX - случайное число
;-------------------------------------------
```
do_Random_EAX rmin:dword,rmax:dword

Неплохой генератор случайного числа получился, вообще сам код взял от сюда:Assembler | Blog. Just Blog

```
;_____________________________________________________________________
[*];Генерация фейковых инструкций ;
;Соглашение о вызовах:
```
cdecl ;

[*]_____
do_fake_instr

Как происходит генерация фейковых интсрукций:

В документации на процессор, можно найти опкоды ассемблерных команд, я использую такие (В будущем можно увеличить):
Code:

```
regw1   db 03h, 0C0h ;add reg1, reg2
regw2 db 2Bh, 0C0h ;sub reg1, reg2
regw3 db 33h, 0C0h ;xor reg1, reg2
regw4 db 8Bh, 0C0h ;mov reg1, reg2
regw5 db 87h, 0C0h ;xchg reg1, reg2
regw6 db 0Bh, 0C0h ;or reg1, reg2
regw7 db 23h, 0C0h ;and reg1, reg2
regw8 db 0F7h, 0D0h ;not reg1
regw9 db 0D1h, 0E0h ;shl reg1, 1
regw10 db 0D1h, 0E8h ;shr reg1, 1
regw11 db 081h, 0E8h ;sub reg1, rnd
regw13 db 081h, 0F0h ;xor reg1, rnd
regw14 db 081h, 0C8h ;or reg1, rnd
regw15 db 081h, 0E0h ;and reg1, rnd
regw16 db 0F7h, 0D8h ;neg reg1
regw17 db 0D1h, 0C0h ;rol reg1, 1
regw18 db 0D1h, 0C8h ;ror reg1, 1
regw19 db 08Dh, 00h ;lea reg1, [reg2]
regd1 db 0B8h; mov reg1, rnd
```

Далее были написаны функции для генерации каждой инструкции, описание всех приводить смысла нет, это можно глянуть в исходники ("fake_instructions.asm"), опишу одну из них:
Code:

```
proc make_xchgreg
mov esi,regw5 ;указатель на xchg reg1, reg2
lodsw ;читаем 2 байта
xor ebx, ebx ;обнуляем ebx для работы
mov ebx, ecx ;ebx=ecx
shl ebx, 3 ;сдвигаем ebx влево на три бита
or ebx, edx ;устанавливем три последних бита из edx
add ah, bl ;добавляем регистры к опкоду
stosw ;сохраняем
ret ;выходим
endp
```

Далее вызываем эти процедуры в случайном порядке в функции do_fake_instr.

2)Вызовы API можно сделать на си, так проще и безопасней, я это сделал так:

Code:

```
static void fake_api_calls(void) {
//Рандомное число, для вызова API в случайном порядке
eax_random = do_Random_EAX(0, 9);

//Рандомно вызываем API
if (eax_random == 0) {
LPSTR wiapi1 = GetCommandLineA();
}
else if (eax_random == 1) {
DWORD wiapi2 = GetTickCount();
}
else if (eax_random == 2) {
DWORD wiapi3 = GetTickCount();
}
else if (eax_random == 3) {
DWORD wiapi4 = GetLastError();
}
else if (eax_random == 4) {
DWORD wiapi5 = GetVersion();
}
else if (eax_random == 5) {
HANDLE wiapi6 = GetCurrentProcess();
}
else if (eax_random == 6) {
HANDLE wiapi7 = GetProcessHeap();
}
else if (eax_random == 7) {
LPWCH wiapi8 = GetEnvironmentStrings();
}
else if (eax_random == 8) {
HANDLE wiapi9 = GetProcessHeap();
}
else if (eax_random == 9) {
LANGID wiapi10 = GetSystemDefaultLangID();
}
}
```

3)В итоге основной модуль движка "fake_api.cpp", в которм интерфейсная функция:
Code:

```
void fake_api_instruction_gen(uint32_t instruction, uint32_t api) {
//Генерация случайных иснтрукций, нужного числа
for (uint32_t i = 0; i < instruction; i++) {
do_fake_instr();
}

//Вызовы случайных API, нужного числа
for (uint32_t i = 0; i < api; i++) {
fake_api_calls();
}
}
```

Описание параметров:

uint32_t instruction - Число интсрукций, которые нужно сгенерировать. uint32_t api - Число API, которые мы вызываем.

4)Пример использования example.cpp:

Code:

```
int main() {
fake_api_instruction_gen(5, 2);
}
```

Сгенерирует 5-ть инструкций и вызовит 2-ве API в случайном порядке, причем порядок будет разный даже при кждом запуске программы...)

Можно запускать в цикле с разными параметрами, я проверил на 10000 инструкций и 5000 вызово API, ничего не крашилось.

Данный движок будет поддерживаться и обновляться в будущем.

Из ближайшего, сделать для x64.

Исходники:XShar/simple_trashe_gen_module

Автор: X-Shar
КИДАЛА