Дизассемблирование

Что это

Дизассемблирование — это процесс преобразования машинного кода в читаемый человеком ассемблерный код.

Что это

По сути, дизассемблер является полной противоположностью ассемблеру. Если Ассемблер конвертирует код написанный на языке ассемблера в двоичный машинный код, то дизассемблер обращает этот процесс и пытается воссоздать код ассемблера из машинного кода.

```
000001D0:  2E 74 65 78-74 00 00 00-CA 65 00 00-00 10 00 00  .text   ▼e   ►
000001E0:  00 66 00 00-00 06 00 00-00 00 00 00-00 00 00 00   f   ♠
000001F0:  00 00 00 00-20 00 00 60-2E 64 61 74-61 00 00 00         '.data
00000200:  44 19 00 00-00 80 00 00-00 06 00 00-00 6C 00 00  D↓   И   ♠   l
00000210:  00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 C0               @  ▼
00000220:  2E 72 73 72-63 00 00 00-00 60 00 00-00 A0 00 00  .rsrc    '   a
00000230:  00 54 00 00-00 72 00 00-00 00 00 00-00 00 00 00   T   r
00000240:  00 00 00 00-40 00 00 40-4D 22 D1 38-48 00 00 00      @  @M"▼8H
00000250:  46 22 D1 38-55 00 00 00-27 C2 F2 37-61 00 00 00  F"▼8U   '▼▼7a
00000260:  34 D0 44 38-6C 00 00 00-34 D0 44 38-79 00 00 00  4▼D8l   4▼D8y
00000270:  84 D3 2B 38-86 00 00 00-34 D0 44 38-90 00 00 00  Д▼+8Ж   4▼D8▼
00000280:  46 22 D1 38-9B 00 00 00-00 00 00 00-00 00 00 00  F"▼8Ы
00000290:  63 6F 6D 64-6C 67 33 32-2E 64 6C 6C-00 53 48 45  comdlg32.dll SHE
000002A0:  4C 4C 33 32-2E 64 6C 6C-00 4D 53 56-43 52 54 2E  LL32.dll MSVCRT.
000002B0:  64 6C 6C 00-41 44 56 41-50 49 33 32-2E 64 6C 6C  dll ADVAPI32.dll
000002C0:  00 4B 45 52-4E 45 4C 33-32 2E 64 6C-6C 00 47 44   KERNEL32.dll GD
000002D0:  49 33 32 2E-64 6C 6C 00-55 53 45 52-33 32 2E 64  I32.dll USER32.d
000002E0:  6C 6C 00 57-49 4E 53 50-4F 4F 4C 2E-44 52 56 00  ll WINSPOOL.DRV
000002F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000300:  EB 02 EB 05-E8 F9 FF FF-FF 58 83 C0-1B 8D A0 01  ▼☻▼╪▼▼▼▼▼X▼▼←▼a☺
00000310:  FC FF FF 83-E4 FC 8B EC-33 C9 66 B9-8F 01 80 30  ▼▼▼▼▼▼Л▼3▼f▼▼☺И0
00000320:  00 40 E2 FA-E8 60 00 00-00 47 65 74-50 72 6F 63   @▼▼▼'   GetProc
00000330:  41 64 64 72-65 73 73 00-4C 6F 61 64-4C 69 62 72  Address LoadLibr
00000340:  61 72 79 41-00 43 72 65-61 74 65 50-72 6F 63 65  aryA CreateProce
00000350:  73 73 41 00-45 78 69 74-50 72 6F 63-65 73 73 00  ssA ExitProcess
00000360:  77 73 32 5F-33 32 00 57-53 41 53 6F-63 6B 65 74  ws2_32 WSASocket
00000370:  41 00 62 69-6E 64 00 6C-69 73 74 65-6E 00 61 63  A bind listen ac
00000380:  63 65 70 74-00 63 6D 64-00 5A 52 BB-00 00 F0 77  cept cmd ZR╗  ▼w
00000390:  81 3B 4D 5A-90 00 74 03-4B EB F5 8B-73 3C 03 F3  ▼;MZ▼ t♥K▼▼Лs<♥▼
000003A0:  8B 76 78 03-F3 8B 7E 20-03 FB 8B 4E-14 56 33 C0  Лvx♥▼Л~ ♥▼ЛN¶V3▼
000003B0:  57 51 8B 3F-03 FB 8B F2-33 C9 B1 0E-F3 A6 59 5F  WQЛ?♥▼Л▼3▼▒♬▼жY_
000003C0:  74 08 83 C7-04 40 E2 E8-FF E1 5E 8B-56 24 03 D3  t◘▼▼♦@▼▼▼^ЛV$♥▼
```

Машинный код	Ассемблер
0E	PUSH CS
1F	POP DS
BA0E00	MOV DX,000E
B409	MOV AH,09
CD21	INT 21
B8014C	MOV AX,4C01
CD21	INT 21
54	PUSH SP
68	DB 68
69	DB 69
7320	JNB 0033
7072	JO 0087
6F	DB 6F
67	DB 67
7261	JB 007A
6D	DB 6D
206361	AND [BP+DI+61],AH
6E	DB 6E
6E	DB 6E
6F	DB 6F

и т.д.

Что это

После компиляции программы, язык на котором она была написана преобразуется в машинный код, но в обратную сторону это уже не работает.

Для чего нужны

Дизассемблеры могут использоваться для обхода мер защиты программы, таких как шифрование или защита от копирования. В связи с этим, они могут быть использованы для злонамеренных целей, таких как распространение вредоносного ПО или кража интеллектуальной собственности.

Для чего нужны

Дизассемблеры могут быть использованы для анализа уязвимостей в программном коде. Это может помочь разработчикам улучшить безопасность программы путем устранения уязвимостей.

Для чего нужны

Дизассемблеры могут быть использованы для работы с программами, для которых нет доступа к исходному коду. Это может помочь разработчикам понять, как работает программа и какие функции в ней используются.

Для чего нужны

Дизассемблеры могут быть использованы для отладки программного кода. Они позволяют разработчику просматривать и изменять значение регистров процессора, память и другие параметры программы во время ее выполнения.

Для чего нужны

Дизассемблеры могут быть использованы для поддержки старых программ, написанных на устаревших языках программирования. Это позволяет сохранить работоспособность этих программ, даже если исходный код больше не доступен.

Обратная разработка

(reverse engineering) - это процесс анализа программного обеспечения с целью понимания его работы и структуры, а также выявления слабых мест, ошибок и уязвимостей.

Обратная разработка

Обратная разработка может быть использована для поиска ошибок и уязвимостей в программном обеспечении, что позволяет повысить его безопасность. Также обратная разработка может быть использована для создания нового продукта на основе уже существующего.

Обратная разработка

Обратная разработка может использоваться и в незаконных целях, таких как создание копий программного обеспечения или взлом программ. Поэтому некоторые страны и компании принимают меры для защиты своих программ от обратной разработки.

Машинный код программы

- это последовательность инструкций, которые процессор может понимать и выполнять непосредственно. Каждая инструкция машинного кода является двоичным числом, которое представляет операцию и данные, над которыми эта операция должна быть выполнена.

Машинный код программы

Машинный код генерируется компилятором из исходного кода программы, и представляет собой непосредственно исполняемый файл программы. Процессор, работая с машинным кодом, выполняет все инструкции по порядку, изменяя состояние памяти и регистров, и получая результаты выполнения программы.

Виды дизассемблеров

- Простые
- С декомпиляцией
- С отладкой
- С автоматической обработкой

Простые дизассемблеры

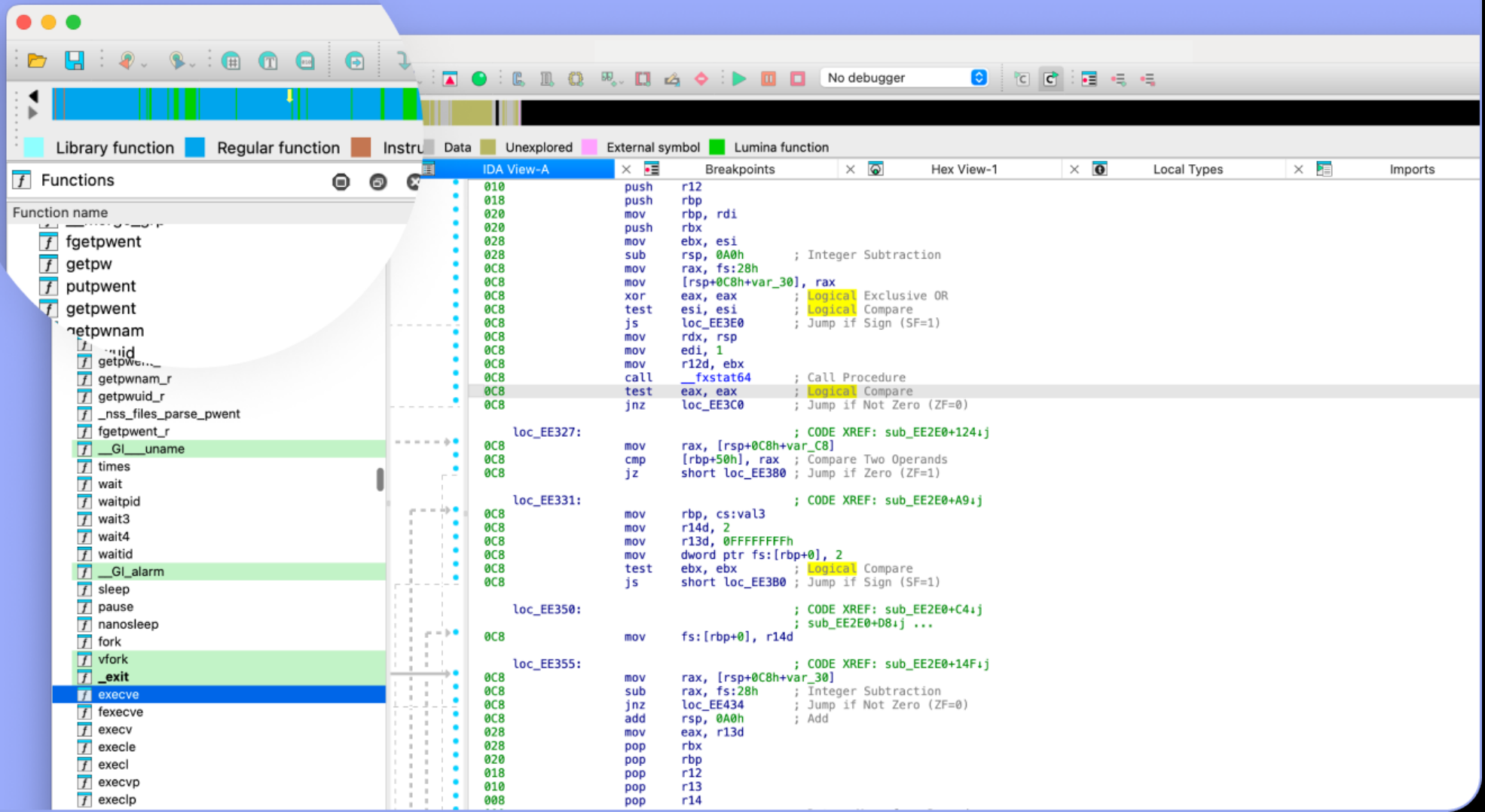
- это программы, которые могут преобразовывать машинный код программы обратно в ассемблерный код. Они могут быть использованы для анализа исполняемых файлов, динамических библиотек и других бинарных файлов.

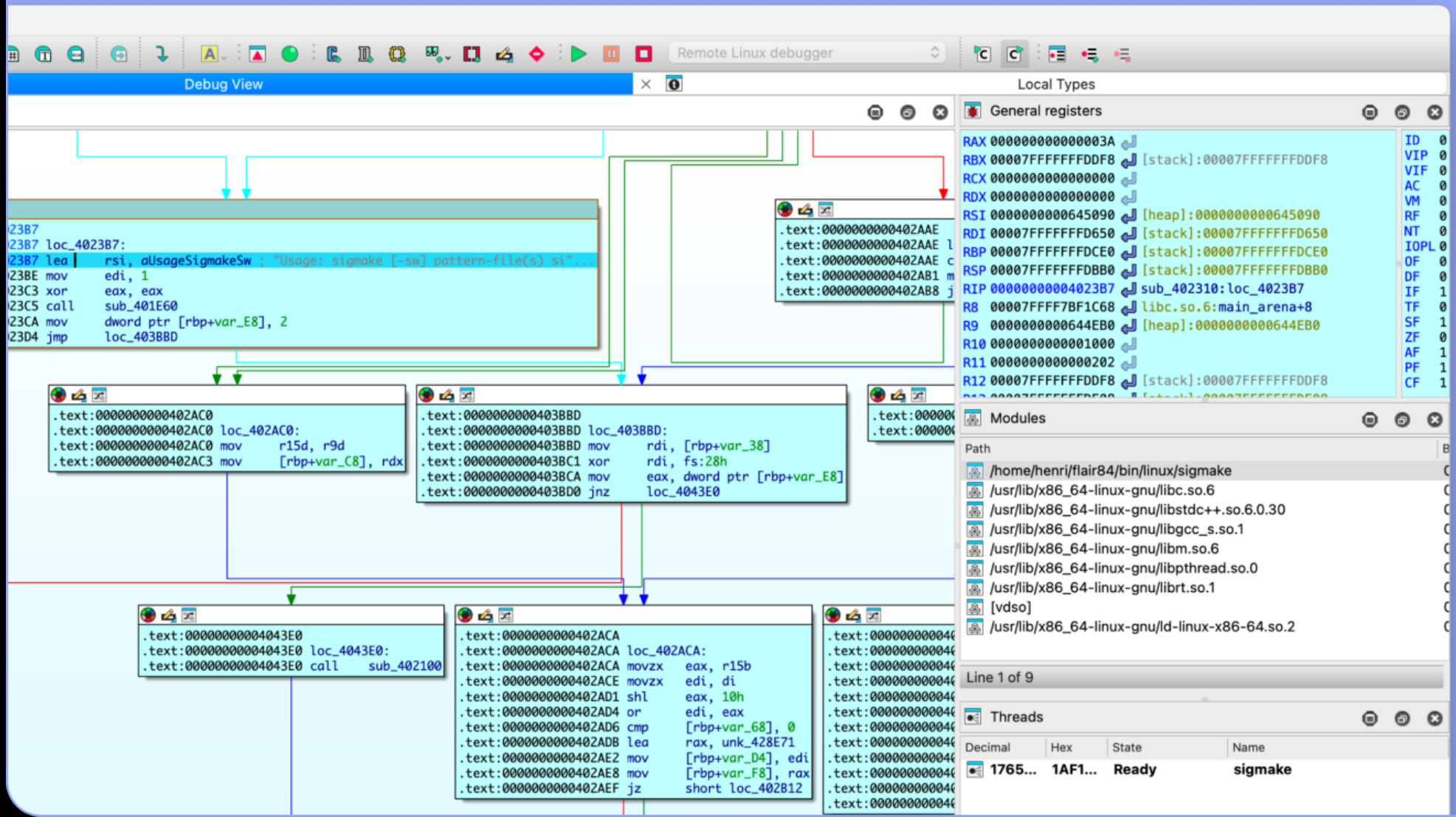
Простые дизассемблеры

Простые дизассемблеры могут быть полезны для начинающих аналитиков и разработчиков, которые хотят изучить программный код и понять, как он работает. Они могут также использоваться для решения конкретных задач, например, поиска конкретной функции в исполняемом файле. Однако для более сложных задач может потребоваться использование более продвинутых инструментов

Простые дизассемблеры

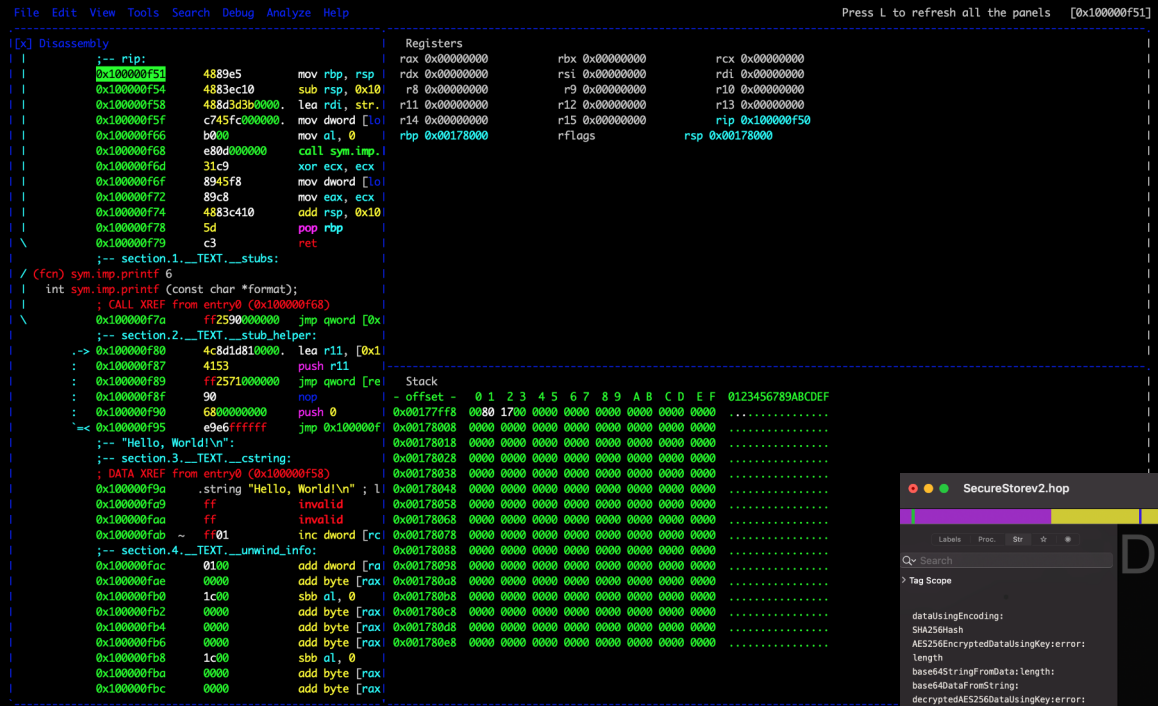
Примером простого дизассемблера может быть программное обеспечение IDA (Interactive Disassembler). IDA является одним из самых популярных и широко используемых дизассемблеров. Он имеет интерфейс с графическими возможностями и может работать с большим количеством форматов исполняемых файлов.

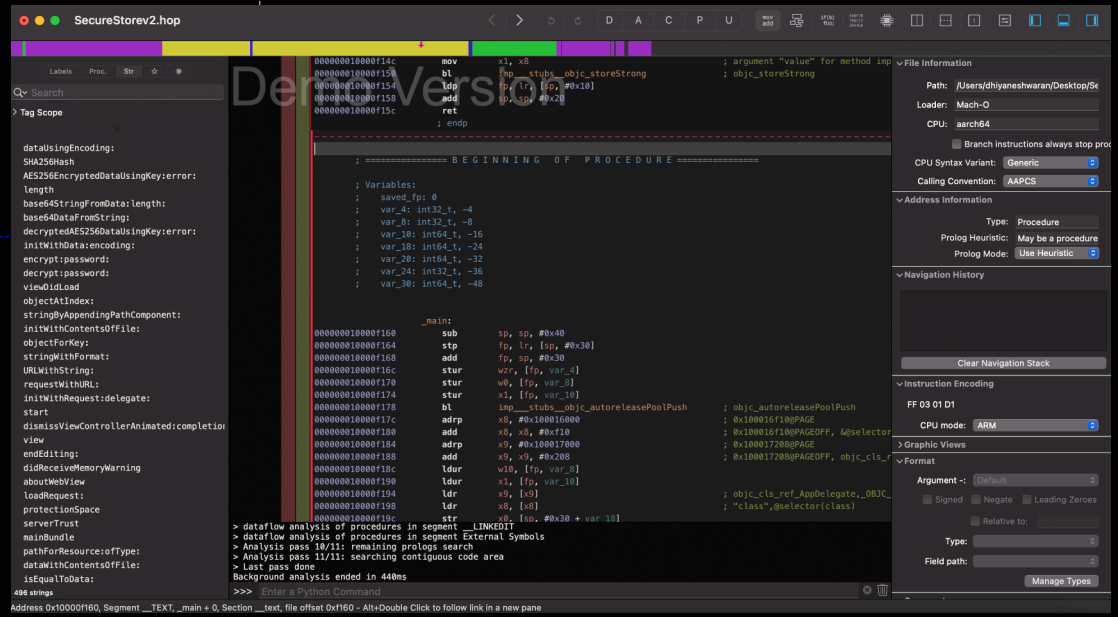

No debugger
Library function
Regular function
Data
Unexplored
External symbol
Lumina function
Functions
Function name
fgetpwent
getpw
putpwent
getpwent
getpwnam
getpwnam_r
getpwuid_r
_nss_files_parse_pwent
fgetpwent_r
__GI___uname
times
wait
waitpid
wait3
wait4
waitid
__GI_alarm
sleep
pause
nanosleep
fork
vfork
_exit
execve
fexecve
execv
execle
execl
execvp
execlp
IDA View-A
Breakpoints
Hex View-1
Local Types
Imports
push r12
push rbp
mov rbp, rdi
push rbx
mov ebx, esi
sub rsp, 0A0h ; Integer Subtraction
mov rax, fs:28h
mov [rsp+0C8h+var_30], rax
xor eax, eax ; Logical Exclusive OR
test esi, esi ; Logical Compare
js loc_EE3E0 ; Jump if Sign (SF=1)
mov rdx, rsp
mov edi, 1
mov r12d, ebx
call __fxstat64 ; Call Procedure
test eax, eax ; Logical Compare
jnz loc_EE3C0 ; Jump if Not Zero (ZF=0)
loc_EE327: ; CODE XREF: sub_EE2E0+124↓j
mov rax, [rsp+0C8h+var_C8]
cmp [rbp+50h], rax ; Compare Two Operands
jz short loc_EE380 ; Jump if Zero (ZF=1)
loc_EE331: ; CODE XREF: sub_EE2E0+A9↓j
mov rbp, cs:val3
mov r14d, 2
mov r13d, 0FFFFFFFFh
mov dword ptr fs:[rbp+0], 2
test ebx, ebx ; Logical Compare
js short loc_EE3B0 ; Jump if Sign (SF=1)
loc_EE350: ; CODE XREF: sub_EE2E0+C4↓j
; sub_EE2E0+D8↓j ...
mov fs:[rbp+0], r14d
loc_EE355: ; CODE XREF: sub_EE2E0+14F↓j
mov rax, [rsp+0C8h+var_30]
sub rax, fs:28h ; Integer Subtraction
jnz loc_EE434 ; Jump if Not Zero (ZF=0)
add rsp, 0A0h ; Add
mov eax, r13d
pop rbx
pop rbp
pop r12
pop r13
pop r14


Remote Linux debugger
Debug View
Local Types
General registers
RAX 000000000000003A
RBX 00007FFFFFFFDDF8 [stack]:00007FFFFFFFDDF8
RCX 0000000000000000
RDX 0000000000000000
RSI 0000000000645090 [heap]:0000000000645090
RDI 00007FFFFFFFD650 [stack]:00007FFFFFFFD650
RBP 00007FFFFFFFDCE0 [stack]:00007FFFFFFFDCE0
RSP 00007FFFFFFFDBB0 [stack]:00007FFFFFFFDBB0
RIP 00000000004023B7 sub_402310:loc_4023B7
R8 00007FFFF7BF1C68 libc.so.6:main_arena+8
R9 0000000000644EB0 [heap]:0000000000644EB0
R10 0000000000001000
R11 0000000000000202
R12 00007FFFFFFFDDF8 [stack]:00007FFFFFFFDDF8
ID 0
VIP 0
VIF 0
AC 0
VM 0
RF 0
NT 0
IOPL 0
OF 0
DF 0
IF 1
TF 0
SF 1
ZF 0
AF 1
PF 1
CF 1
Modules
Path
/home/henri/flair84/bin/linux/sigmake
/usr/lib/x86_64-linux-gnu/libc.so.6
/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.30
/usr/lib/x86_64-linux-gnu/libgcc_s.so.1
/usr/lib/x86_64-linux-gnu/libm.so.6
/usr/lib/x86_64-linux-gnu/libpthread.so.0
/usr/lib/x86_64-linux-gnu/librt.so.1
[vdso]
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
Line 1 of 9
Threads
Decimal
Hex
State
Name
1765...
1AF1...
Ready
sigmake
23B7
23B7 loc_4023B7:
23B7 lea rsi, aUsageSigmakeSw ; "Usage: sigmake [-sw] pattern-file(s) si"...
23BE mov edi, 1
23C3 xor eax, eax
23C5 call sub_401E60
23CA mov dword ptr [rbp+var_E8], 2
23D4 jmp loc_403BBD
.text:0000000000402AAE
.text:0000000000402AAE l
.text:0000000000402AAE c
.text:0000000000402AB1 m
.text:0000000000402AB8 j
.text:0000000000402AC0
.text:0000000000402AC0 loc_402AC0:
.text:0000000000402AC0 mov r15d, r9d
.text:0000000000402AC3 mov [rbp+var_C8], rdx
.text:0000000000403BBD
.text:0000000000403BBD loc_403BBD:
.text:0000000000403BBD mov rdi, [rbp+var_38]
.text:0000000000403BC1 xor rdi, fs:28h
.text:0000000000403BCA mov eax, dword ptr [rbp+var_E8]
.text:0000000000403BD0 jnz loc_4043E0
.text:00000000004043E0
.text:00000000004043E0 loc_4043E0:
.text:00000000004043E0 call sub_402100
.text:0000000000402ACA
.text:0000000000402ACA loc_402ACA:
.text:0000000000402ACA movzx eax, r15b
.text:0000000000402ACE movzx edi, di
.text:0000000000402AD1 shl eax, 10h
.text:0000000000402AD4 or edi, eax
.text:0000000000402AD6 cmp [rbp+var_68], 0
.text:0000000000402ADB lea rax, unk_428E71
.text:0000000000402AE2 mov [rbp+var_D4], edi
.text:0000000000402AE8 mov [rbp+var_F8], rax
.text:0000000000402AEF jz short loc_402B12

Простые дизассемблеры

IDA является платным программным обеспечением, однако существуют и бесплатные альтернативы, например, Radare2 или Hopper Disassembler. Они также предоставляют возможность дизассемблирования и анализа исполняемых файлов, но имеют более ограниченные возможности, чем IDA.


File Edit View Tools Search Debug Analyze Help
Press L to refresh all the panels [0x100000f51]
[x] Disassembly
;-- rip:
0x100000f51 4889e5 mov rbp, rsp
0x100000f54 4883ec10 sub rsp, 0x10
0x100000f58 488d3d3b0000. lea rdi, str.
0x100000f5f c745fc000000. mov dword [lo
0x100000f66 b000 mov al, 0
0x100000f68 e80d000000 call sym.imp.
0x100000f6d 31c9 xor ecx, ecx
0x100000f6f 8945f8 mov dword [lo
0x100000f72 89c8 mov eax, ecx
0x100000f74 4883c410 add rsp, 0x10
0x100000f78 5d pop rbp
0x100000f79 c3 ret
;-- section.1.__TEXT.__stubs:
/ (fcn) sym.imp.printf 6
int sym.imp.printf (const char *format);
; CALL XREF from entry0 (0x100000f68)
0x100000f7a ff2590000000 jmp qword [0x
;-- section.2.__TEXT.__stub_helper:
.-> 0x100000f80 4c8d1d810000. lea r11, [0x1
: 0x100000f87 4153 push r11
: 0x100000f89 ff2571000000 jmp qword [re
: 0x100000f8f 90 nop
: 0x100000f90 6800000000 push 0
`=< 0x100000f95 e9e6ffffff jmp 0x100000f
;-- "Hello, World!\n":
;-- section.3.__TEXT.__cstring:
; DATA XREF from entry0 (0x100000f58)
0x100000f9a .string "Hello, World!\n" ; l
0x100000fa9 ff invalid
0x100000faa ff invalid
0x100000fab ~ ff01 inc dword [rc
;-- section.4.__TEXT.__unwind_info:
0x100000fac 0100 add dword [ra
0x100000fae 0000 add byte [rax
0x100000fb0 1c00 sbb al, 0
0x100000fb2 0000 add byte [rax
0x100000fb4 0000 add byte [rax
0x100000fb6 0000 add byte [rax
0x100000fb8 1c00 sbb al, 0
0x100000fba 0000 add byte [rax
0x100000fbc 0000 add byte [rax
Registers
rax 0x00000000 rbx 0x00000000 rcx 0x00000000
rdx 0x00000000 rsi 0x00000000 rdi 0x00000000
r8 0x00000000 r9 0x00000000 r10 0x00000000
r11 0x00000000 r12 0x00000000 r13 0x00000000
r14 0x00000000 r15 0x00000000 rip 0x100000f50
rbp 0x00178000 rflags rsp 0x00178000
Stack
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00177ff8 0080 1700 0000 0000 0000 0000 0000 0000 ....
0x00178008 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178018 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178028 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178038 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178048 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178058 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178068 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178078 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178088 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x00178098 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x001780a8 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x001780b8 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x001780c8 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x001780d8 0000 0000 0000 0000 0000 0000 0000 0000 ....
0x001780e8 0000 0000 0000 0000 0000 0000 0000 0000 ....


SecureStorev2.hop
Demo Version
Labels Proc. Str
Search
Tag Scope
dataUsingEncoding:
SHA256Hash
AES256EncryptedDataUsingKey:error:
length
base64StringFromData:length:
base64DataFromString:
decryptedAES256DataUsingKey:error:
initWithData:encoding:
encrypt:password:
decrypt:password:
viewOidLoad
objectAtIndex:
stringByAppendingPathComponent:
initWithContentsOfFile:
objectForKey:
stringWithFormat:
URLWithString:
requestWithURL:
initWithRequest:delegate:
start
dismissViewControllerAnimated:completio
view
endEditing:
didReceiveMemoryWarning
aboutWebView
loadRequest:
protectionSpace
serverTrust
mainBundle
pathForResource:ofType:
dataWithContentsOfFile:
isEqualToData:
496 strings
; ================ B E G I N N I N G O F P R O C E D U R E ================
; Variables:
; saved_fp: 0
; var_4: int32_t, -4
; var_8: int32_t, -8
; var_10: int64_t, -16
; var_18: int64_t, -24
; var_20: int64_t, -32
; var_24: int32_t, -36
; var_30: int64_t, -48
_main:
sub sp, sp, #0x40
stp fp, lr, [sp, #0x30]
add fp, sp, #0x30
stur wzr, [fp, var_4]
stur w0, [fp, var_8]
stur x1, [fp, var_10]
bl imp___stubs__objc_autoreleasePoolPush ; objc_autoreleasePoolPush
adrp x8, #0x100016000 ; 0x100016f100@PAGE
add x8, x8, #0xf10 ; 0x100016f100@PAGEOFF, &@selector
adrp x9, #0x100017000 ; 0x100017208@PAGE
add x9, x9, #0x208 ; 0x100017208@PAGEOFF, objc_cls_
ldur w10, [fp, var_8]
ldur x1, [fp, var_10]
ldr x9, [x9] ; objc_cls_ref_AppDelegate, _OBJC_
ldr x8, [x8] ; "class",@selector(class)
str x0, [sp, #0x30 + var_18]
> dataflow analysis of procedures in segment __LINKEDIT
> dataflow analysis of procedures in segment External Symbols
> Analysis pass 10/11: remaining prologs search
> Analysis pass 11/11: searching contiguous code area
> Last pass done
Background analysis ended in 440ms
>>> Enter a Python Command
Address 0x10000f160, Segment __TEXT, _main + 0, Section __text, file offset 0xf160 - Alt+Double Click to follow link in a new pane
File Information
Path: /Users/dhiyaneshwaran/Desktop/Se
Loader: Mach-O
CPU: aarch64
Branch instructions always stop proc
CPU Syntax Variant: Generic
Calling Convention: AAPCS
Address Information
Type: Procedure
Prolog Heuristic: May be a procedure
Prolog Mode: Use Heuristic
Navigation History
Clear Navigation Stack
Instruction Encoding
FF 03 01 D1
CPU mode: ARM
Graphic Views
Format
Argument -: Default
Signed Negate Leading Zeroes
Relative to:
Type:
Field path:
Manage Types

Дизассемблеры с декомпиляцией

- это программное обеспечение, которое помогает восстановить исходный код программы из машинного кода. В отличие от простых дизассемблеров, которые преобразуют машинный код в ассемблерный код, дизассемблеры с декомпиляцией могут восстанавливать более абстрактный уровень исходного кода, например, на языке C или C++.

Дизассемблеры с декомпиляцией

Дизассемблеры с декомпиляцией используют анализ машинного кода программы и поиск схожих паттернов в коде, чтобы восстановить функции и другие элементы исходного кода. Для этого они используют различные методы и алгоритмы, такие как анализ потока управления, статический анализ и т.д. Результатом работы дизассемблера с декомпиляцией является код на более высоком уровне, который проще понимать и анализировать, чем ассемблерный код. https://habr.com/ru/articles/447450/

Дизассемблеры с декомпиляцией

Ghidra — это бесплатный кроссплатформенный интерактивный дизассемблер и декомпилятор с модульной структурой, с поддержкой почти всех основных архитектур ЦПУ и гибким графическим интерфейсом для работы с дизассемблированным кодом, памятью, восстановленным (декомпилированным) кодом, отладочными символами и многое-многое другое.

Program Trees
- stack0.exe
  - .bss
  - .data
  - .got.plt
  - .got
  - .dynamic
  - .jcr
  - .fini_array
  - .init_array
  - .eh_frame
  - .eh_frame_hdr
  - .rodata
  - .fini
  - .text
  - .plt.got
  - .plt
  - .init
  - .rel.plt
  - .rel.dyn

Program Tree

Symbol Tree
- __libc_csu_fini
- __libc_csu_init
- __libc_start_main
- __libc_start_main
- __x86.get_pc_thunk.bx
- _fini
- _init
- _start
- deregister_tm_clones
- frame_dummy
- FUN_080482f0
- gets
- gets
- main
- puts
- puts
- register_tm_clones
- Labels
- Classes
- Namespaces

Filter:

Data Type Manager
- Data Types
  - BuiltInTypes
  - stack0.exe
  - generic_clib
  - generic_clib_64

Filter:

Listing: stack0.exe

stack0.exe

```
08048410 b8 10 9f 04 08   MOV     EAX,__JCR_END__
08048415 8b 10            MOV     EDX,dword ptr [EAX]=>__JCR_END__
08048417 85 d2            TEST    EDX,EDX
08048419 75 05            JNZ     LAB_08048420

                     LAB_0804841b                            XREF[1]:  08048427(j)
0804841b eb 93            JMP     register_tm_clones                undefined register_tm_clo…
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)
0804841d 8d               ??      8Dh
0804841e 76               ??      76h    v
0804841f 00               ??      00h

                     LAB_08048420                            XREF[1]:  08048419(j)
08048420 ba 00 00 00 00   MOV     EDX,0x0
08048425 85 d2            TEST    EDX,EDX
08048427 74 f2            JZ      LAB_0804841b
08048429 55               PUSH    EBP
0804842a 89 e5            MOV     EBP,ESP
0804842c 83 ec 14         SUB     ESP,0x14
0804842f 50               PUSH    EAX=>__JCR_END__
08048430 ff d2            CALL    EDX
08048432 83 c4 10         ADD     ESP,0x10
08048435 c9               LEAVE
08048436 e9 75 ff ff ff   JMP     register_tm_clones                undefined register_tm_clo…
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                     ************************************************
                     *                    FUNCTION                  *
                     ************************************************
                     int __cdecl main(void)
        int            EAX:4      <RETURN>
        undefined4     Stack[0x0]…local_res0                XREF[1]:  08048442(R)
        undefined4     Stack[-0xc…local_c                   XREF[1]:  08048490(R)
        int            Stack[-0x1…modified                  XREF[2]:  0804844c(W),
                                                                      08048462(R)
        undefined1     Stack[-0x5…local_54                  XREF[1]:  08048456(*)
                     main                                   XREF[4]:  Entry Point(*),
                                                                      _start:08048357(*),
                                                                      08048568, 080485c4(*)
0804843b 8d 4c 24 04      LEA     ECX=>Stack[0x4],[ESP + 0x4]
0804843f 83 e4 f0         AND     ESP,0xfffffff0
08048442 ff 71 fc         PUSH    dword ptr [ECX + local_res0]
08048445 55               PUSH    EBP
08048446 89 e5            MOV     EBP,ESP
08048448 51               PUSH    ECX
08048449 83 ec 54         SUB     ESP,0x54
0804844c c7 45 f4 00 00 00 00   MOV     dword ptr [EBP + modified],0x0
08048453 83 ec 0c         SUB     ESP,0xc
08048456 8d 45 b4         LEA     EAX=>local_54,[EBP + -0x4c]
08048459 50               PUSH    EAX
0804845a e8 a1 fe ff ff   CALL    gets                              char * gets(char * __s)
0804845f 83 c4 10         ADD     ESP,0x10
08048462 8b 45 f4         MOV     EAX,dword ptr [EBP + modified]
08048465 85 c0            TEST    EAX,EAX
08048467 74 12            JZ      LAB_0804847b
08048469 83 ec 0c         SUB     ESP,0xc
0804846c 68 20 85 04 08   PUSH    s_you_have_changed_the_'modified'_v_…= "you have changed the '…
```

Overview
Data (.strtab .strtab::00000072)

Decompile: main - (stack0.exe)

```
1
2  int main(void)
3
4  {
5    char local_54 [64];
6    int modified;
7
8    modified = 0;
9    gets(local_54);
10   if (modified == 0) {
11     puts("Try again?");
12   }
13   else {
14     puts("you have changed the \'modified\' variable");
15   }
16   return 0;
17 }
18
```

Decompile: main | Defined Strings

==Дизассемблеры с отладкой==

- это программное обеспечение, которое позволяет производить отладку исполняемых файлов, используя дизассемблированный код вместо исходного. Они позволяют анализировать работу программы в процессе выполнения и искать ошибки в машинном коде.

==Дизассемблеры с отладкой==

- это программное обеспечение, которое позволяет производить отладку исполняемых файлов, используя дизассемблированный код вместо исходного. Они позволяют анализировать работу программы в процессе выполнения и искать ошибки в машинном коде.

Дизассемблеры с отладкой

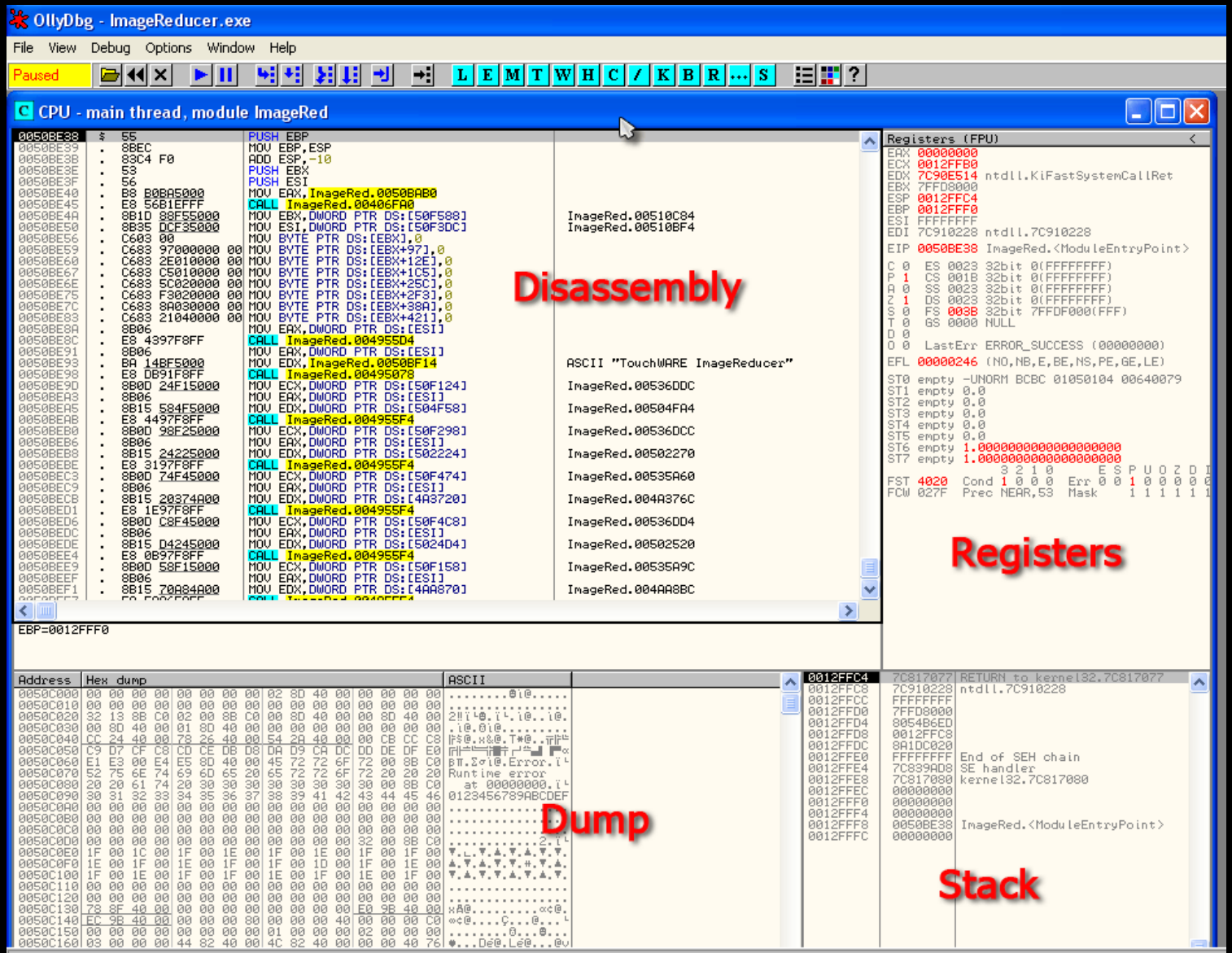
Для работы с дизассемблером с отладкой необходимо иметь исполняемый файл программы, которую нужно отлаживать. Дизассемблер с отладкой загружает исполняемый файл в память и начинает отслеживать его работу. Пользователь может установить точки останова (breakpoints) в коде, чтобы остановить выполнение программы в определенных местах и проанализировать текущее состояние системы.

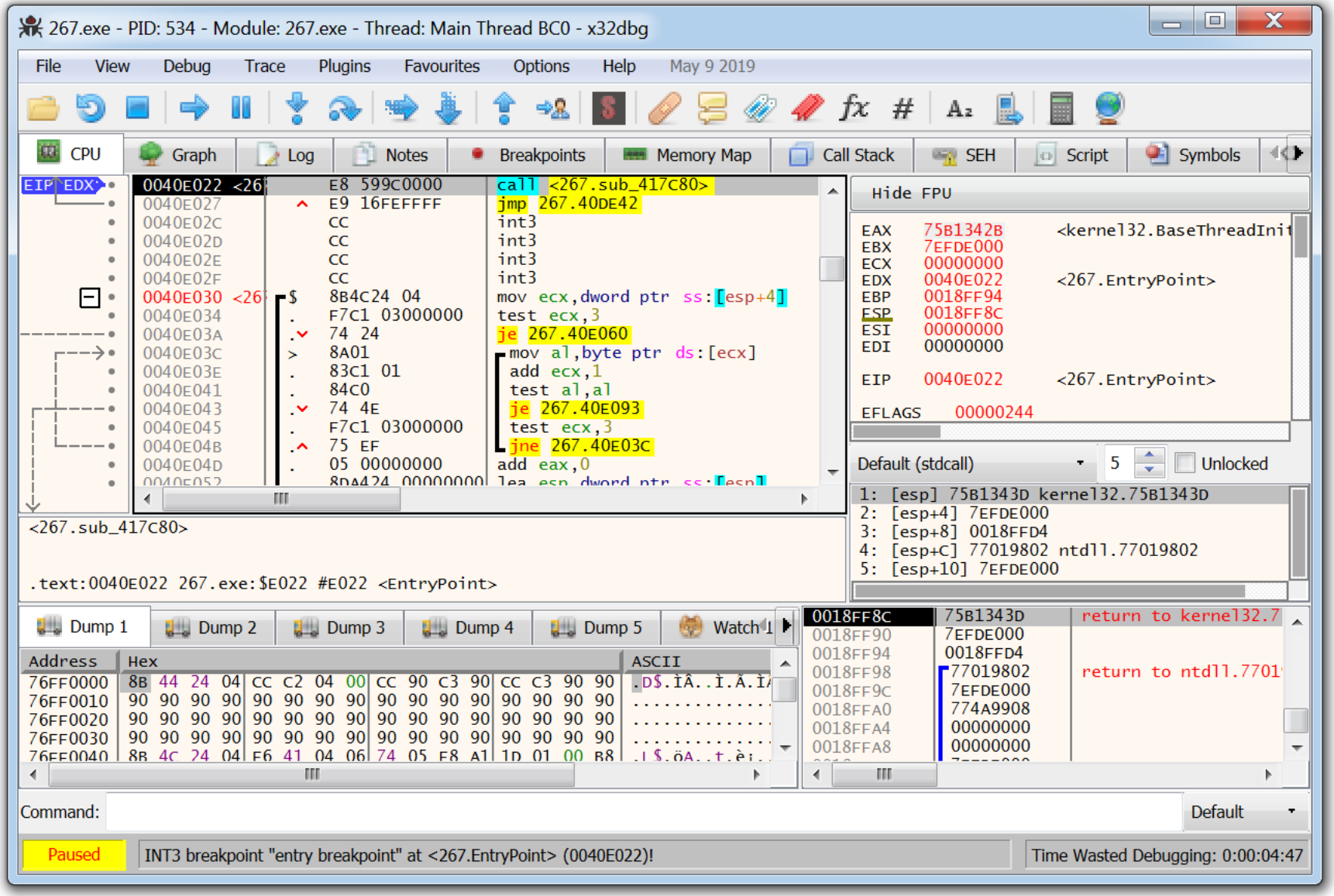
Дизассемблеры с отладкой

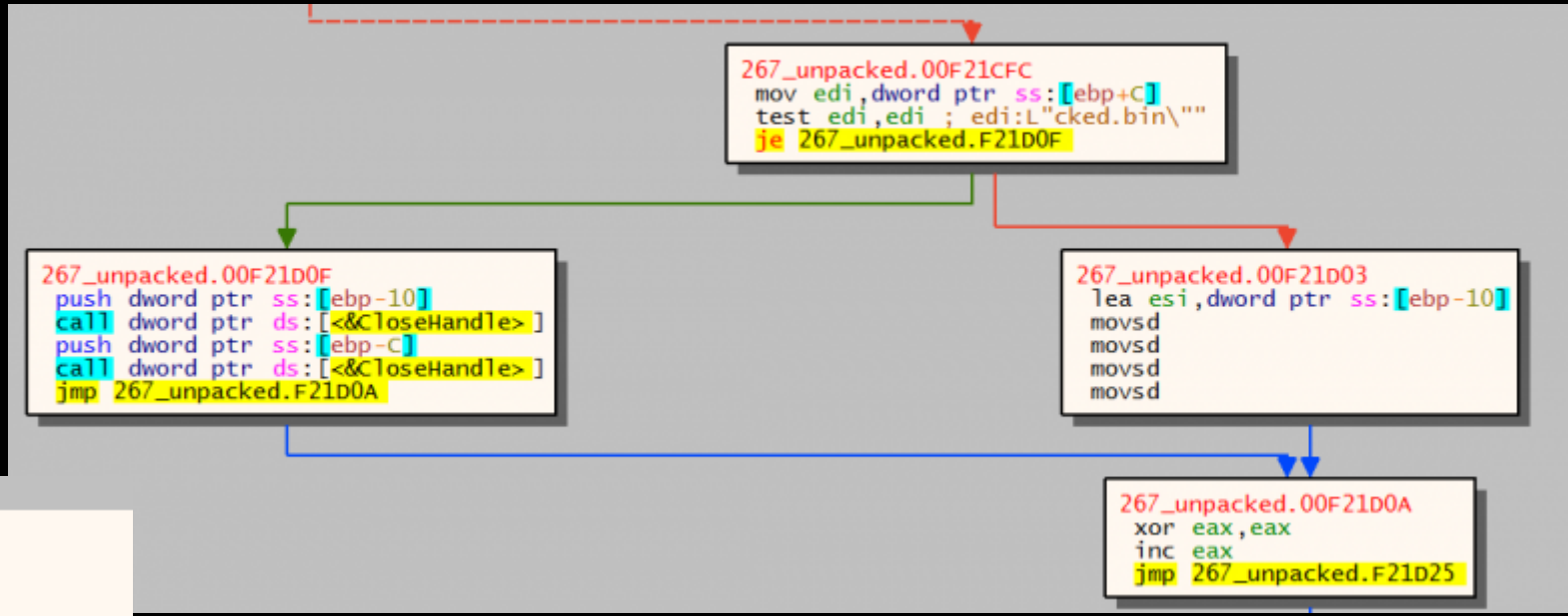
При остановке программы на точке останова пользователь может просмотреть состояние регистров процессора, стека, памяти и других ресурсов. Дизассемблер с отладкой также позволяет выполнить код пошагово, что может помочь понять, как работает программа и где возникла ошибка. Он также может предоставить информацию о вызовах функций и параметрах, передаваемых в них.

Дизассемблеры с отладкой

Примеры дизассемблеров с отладкой включают в себя OllyDbg, Immunity Debugger, x64dbg, GDB и другие. Они могут использоваться для отладки исполняемых файлов, драйверов, системных библиотек и других программных компонентов.


OllyDbg - ImageReducer.exe
File View Debug Options Window Help
Paused
CPU - main thread, module ImageRed
0050BE38 $ 55 PUSH EBP
0050BE39 . 8BEC MOV EBP,ESP
0050BE3B . 83C4 F0 ADD ESP,-10
0050BE3E . 53 PUSH EBX
0050BE3F . 56 PUSH ESI
0050BE40 . B8 B0BA5000 MOV EAX,ImageRed.0050BAB0
0050BE45 . E8 56B1EFFF CALL ImageRed.00406FA0
0050BE4A . 8B1D 88F55000 MOV EBX,DWORD PTR DS:[50F588] ImageRed.00510C84
0050BE50 . 8B35 DCF35000 MOV ESI,DWORD PTR DS:[50F3DC] ImageRed.00510BF4
0050BE56 . C603 00 MOV BYTE PTR DS:[EBX],0
0050BE59 . C683 97000000 00 MOV BYTE PTR DS:[EBX+97],0
0050BE60 . C683 2E010000 00 MOV BYTE PTR DS:[EBX+12E],0
0050BE67 . C683 C5010000 00 MOV BYTE PTR DS:[EBX+1C5],0
0050BE6E . C683 5C020000 00 MOV BYTE PTR DS:[EBX+25C],0
0050BE75 . C683 F3020000 00 MOV BYTE PTR DS:[EBX+2F3],0
0050BE7C . C683 8A030000 00 MOV BYTE PTR DS:[EBX+38A],0
0050BE83 . C683 21040000 00 MOV BYTE PTR DS:[EBX+421],0
0050BE8A . 8B06 MOV EAX,DWORD PTR DS:[ESI]
0050BE8C . E8 4397F8FF CALL ImageRed.004955D4
0050BE91 . 8B06 MOV EAX,DWORD PTR DS:[ESI]
0050BE93 . BA 14BF5000 MOV EDX,ImageRed.0050BF14 ASCII "TouchWARE ImageReducer"
0050BE98 . E8 DB91F8FF CALL ImageRed.00495078
0050BE9D . 8B0D 24F15000 MOV ECX,DWORD PTR DS:[50F124] ImageRed.00536DDC
0050BEA3 . 8B06 MOV EAX,DWORD PTR DS:[ESI]
0050BEA5 . 8B15 584F5000 MOV EDX,DWORD PTR DS:[504F58] ImageRed.00504FA4
0050BEAB . E8 4497F8FF CALL ImageRed.004955F4
0050BEB0 . 8B0D 98F25000 MOV ECX,DWORD PTR DS:[50F298] ImageRed.00536DCC
0050BEB6 . 8B06 MOV EAX,DWORD PTR DS:[ESI]
0050BEB8 . 8B15 24225000 MOV EDX,DWORD PTR DS:[502224] ImageRed.00502270
0050BEBE . E8 3197F8FF CALL ImageRed.004955F4
0050BEC3 . 8B0D 74F45000 MOV ECX,DWORD PTR DS:[50F474] ImageRed.00535A60
0050BEC9 . 8B06 MOV EAX,DWORD PTR DS:[ESI]
0050BECB . 8B15 20374A00 MOV EDX,DWORD PTR DS:[4A3720] ImageRed.004A376C
0050BED1 . E8 1E97F8FF CALL ImageRed.004955F4
0050BED6 . 8B0D C8F45000 MOV ECX,DWORD PTR DS:[50F4C8] ImageRed.00536DD4
0050BEDC . 8B06 MOV EAX,DWORD PTR DS:[ESI]
0050BEDE . 8B15 D4245000 MOV EDX,DWORD PTR DS:[5024D4] ImageRed.00502520
0050BEE4 . E8 0B97F8FF CALL ImageRed.004955F4
0050BEE9 . 8B0D 58F15000 MOV ECX,DWORD PTR DS:[50F158] ImageRed.00535A9C
0050BEEF . 8B06 MOV EAX,DWORD PTR DS:[ESI]
0050BEF1 . 8B15 70A84A00 MOV EDX,DWORD PTR DS:[4AA870] ImageRed.004AA8BC
Disassembly
EBP=0012FFF0
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228
EIP 0050BE38 ImageRed.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BCBC 01050104 00640079
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
3 2 1 0 E S P U O Z D I
FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
Registers
Address Hex dump ASCII
0050C000 00 00 00 00 00 00 00 00 02 8D 40 00 00 00 00 00 .....@.....
0050C010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0050C020 32 13 8B C0 02 00 8B C0 00 8D 40 00 00 8D 40 00 2.....@...@.
0050C030 00 8D 40 00 01 8D 40 00 00 00 00 00 00 00 00 00 ..@...@.........
0050C040 CC 24 40 00 78 26 40 00 54 2A 40 00 00 CB CC C8 .$@.x&@.T*@.....
0050C050 C9 D7 CF C8 CD CE D8 D8 DA D9 CA DC DD DE DF E0 ....
0050C060 E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0 ......@.Error...
0050C070 52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20 Runtime error
0050C080 20 20 61 74 20 30 30 30 30 30 30 30 30 00 8B C0 at 00000000...
0050C090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDEF
0050C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0050C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0050C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0050C0D0 00 00 00 00 00 00 00 00 00 00 00 00 32 00 8B C0 ....2...
0050C0E0 1F 00 1C 00 1F 00 1E 00 1F 00 1E 00 1F 00 1F 00 ▼.└.▼.▲.▼.▲.▼.▼.
0050C0F0 1E 00 1F 00 1E 00 1F 00 1F 00 1D 00 1F 00 1E 00 ▲.▼.▲.▼.▼.↔.▼.▲.
0050C100 1F 00 1E 00 1F 00 1F 00 1E 00 1F 00 1E 00 1F 00 ▼.▲.▼.▼.▲.▼.▲.▼.
0050C110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0050C120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0050C130 78 8F 40 00 00 00 00 00 00 00 00 00 E0 9B 40 00 x@.........α@.
0050C140 EC 9B 40 00 00 00 00 00 80 00 00 00 40 00 00 C0 ∞@.....Ç...@...└
0050C150 00 00 00 00 00 00 00 00 01 00 00 00 02 00 00 00 ....
0050C160 03 00 00 00 44 82 40 00 4C 82 40 00 00 00 40 76 ♥...D@.L@...@v
Dump
0012FFC4 7C817077 RETURN to kernel32.7C817077
0012FFC8 7C910228 ntdll.7C910228
0012FFCC FFFFFFFF
0012FFD0 7FFD8000
0012FFD4 8054B6ED
0012FFD8 0012FFC8
0012FFDC 8A1DC020
0012FFE0 FFFFFFFF End of SEH chain
0012FFE4 7C839AD8 SE handler
0012FFE8 7C817080 kernel32.7C817080
0012FFEC 00000000
0012FFF0 00000000
0012FFF4 00000000
0012FFF8 0050BE38 ImageRed.<ModuleEntryPoint>
0012FFFC 00000000
Stack
Analysing ImageRed: 5183 heuristical procedures, 2295 calls to known, 1471 calls to guessed functions


267.exe - PID: 534 - Module: 267.exe - Thread: Main Thread BC0 - x32dbg
File View Debug Trace Plugins Favourites Options Help May 9 2019
CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols
EIP EDX
0040E022 <26 E8 599C0000 call <267.sub_417C80>
0040E027 E9 16FEFFFF jmp 267.40DE42
0040E02C CC int3
0040E02D CC int3
0040E02E CC int3
0040E02F CC int3
0040E030 <26 $ 8B4C24 04 mov ecx,dword ptr ss:[esp+4]
0040E034 . F7C1 03000000 test ecx,3
0040E03A . 74 24 je 267.40E060
0040E03C > 8A01 mov al,byte ptr ds:[ecx]
0040E03E . 83C1 01 add ecx,1
0040E041 . 84C0 test al,al
0040E043 . 74 4E je 267.40E093
0040E045 . F7C1 03000000 test ecx,3
0040E04B . 75 EF jne 267.40E03C
0040E04D . 05 00000000 add eax,0
<267.sub_417C80>
.text:0040E022 267.exe:$E022 #E022 <EntryPoint>
Hide FPU
EAX 75B1342B <kernel32.BaseThreadInit
EBX 7EFDE000
ECX 00000000
EDX 0040E022 <267.EntryPoint>
EBP 0018FF94
ESP 0018FF8C
ESI 00000000
EDI 00000000
EIP 0040E022 <267.EntryPoint>
EFLAGS 00000244
Default (stdcall) 5 Unlocked
1: [esp] 75B1343D kernel32.75B1343D
2: [esp+4] 7EFDE000
3: [esp+8] 0018FFD4
4: [esp+C] 77019802 ntdll.77019802
5: [esp+10] 7EFDE000
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1
Address Hex ASCII
76FF0000 8B 44 24 04 CC C2 04 00 CC 90 C3 90 CC C3 90 90 .D$.ÌÂ..Ì.Ã.Ì
76FF0010 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ....
76FF0020 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ....
76FF0030 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ....
76FF0040 8B 4C 24 04 F6 41 04 06 74 05 E8 A1 1D 01 00 B8 .L$.öA..t.è¡...
0018FF8C 75B1343D return to kernel32.7
0018FF90 7EFDE000
0018FF94 0018FFD4
0018FF98 77019802 return to ntdll.7701
0018FF9C 7EFDE000
0018FFA0 774A9908
0018FFA4 00000000
0018FFA8 00000000
Command: Default
Paused
INT3 breakpoint "entry breakpoint" at <267.EntryPoint> (0040E022)!
Time Wasted Debugging: 0:00:04:47




```
push ebp
mov ebp,esp
sub esp,420
push ebx
push esi
push edi
call 267_unpacked.F2AEC9
call 267_unpacked.F2BE17
mov ebx,104
lea eax,dword ptr ss:[ebp-420]
push ebx
push eax
push 0
call dword ptr ds:[F30BD8]
lea ecx,dword ptr ss:[ebp-420]
call 267_unpacked.F21144
```

Дизассемблирование

Нарушение авторских прав

Уязвимости безопасности

Несанкционированный доступ

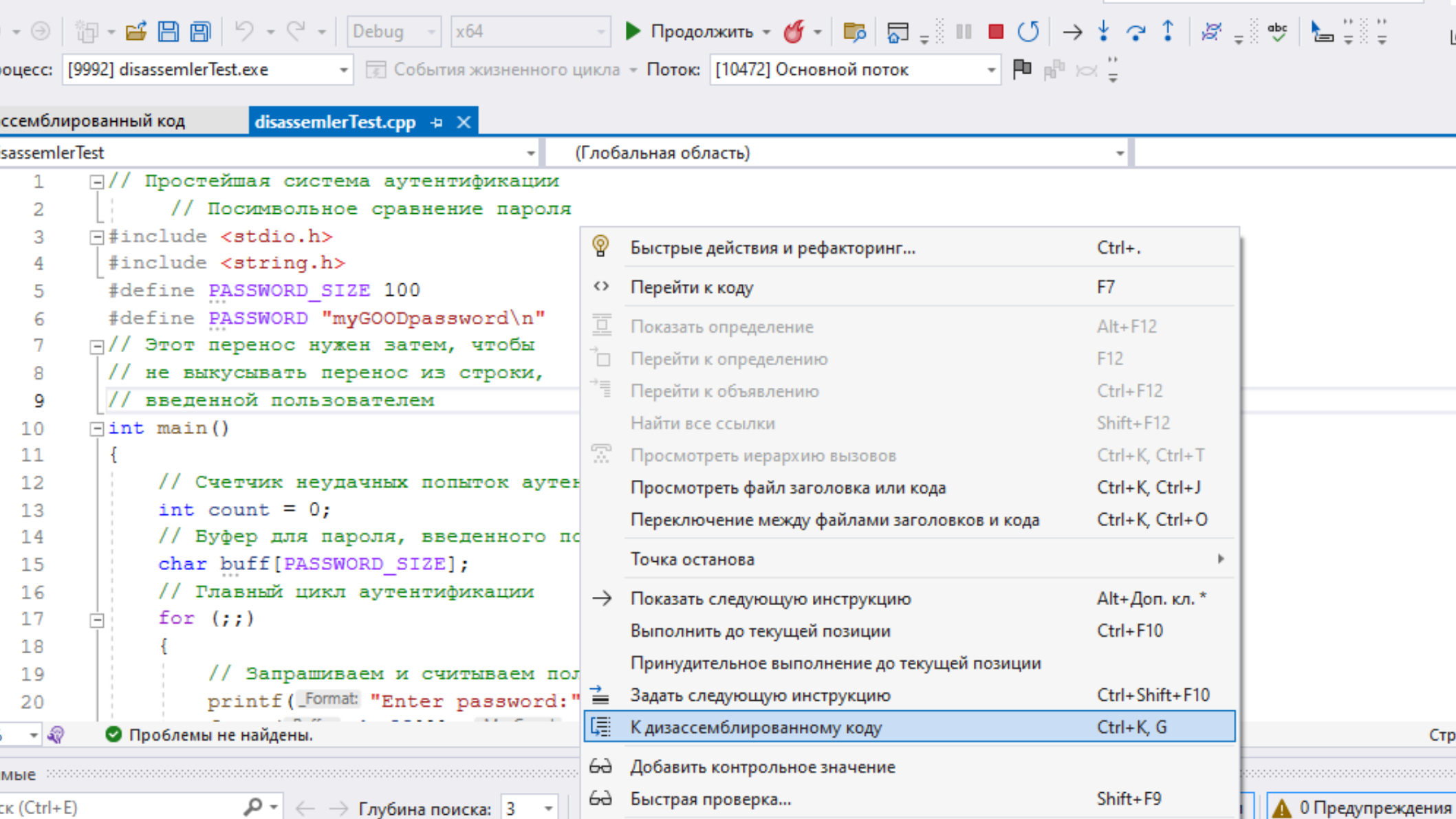
Неправомерное использование

Нарушение договорных обязательств

Антидизассемблинг

Антидизассемблинг — это набор техник, используемых для затруднения или предотвращения корректного дизассемблирования программы. Эти методы применяются в:

- Защите ПО от реверс-инжиниринга (DRM, лицензирование).
- Вредоносном ПО (вирусы, руткиты).
- Обфускации кода (затруднение анализа).


Debug
x64
Продолжить
[9992] disassemlerTest.exe
События жизненного цикла
Поток:
[10472] Основной поток
ссемблированный код
disassemlerTest.cpp
sassemlerTest
(Глобальная область)
// Простейшая система аутентификации
// Посимвольное сравнение пароля
#include <stdio.h>
#include <string.h>
#define PASSWORD_SIZE 100
#define PASSWORD "myGOODpassword\n"
// Этот перенос нужен затем, чтобы
// не выкусывать перенос из строки,
// введенной пользователем
int main()
{
// Счетчик неудачных попыток аутен
int count = 0;
// Буфер для пароля, введенного по
char buff[PASSWORD_SIZE];
// Главный цикл аутентификации
for (;;)
{
// Запрашиваем и считываем пол
printf(Format: "Enter password:"
Быстрые действия и рефакторинг... Ctrl+.
Перейти к коду F7
Показать определение Alt+F12
Перейти к определению F12
Перейти к объявлению Ctrl+F12
Найти все ссылки Shift+F12
Просмотреть иерархию вызовов Ctrl+K, Ctrl+T
Просмотреть файл заголовка или кода Ctrl+K, Ctrl+J
Переключение между файлами заголовков и кода Ctrl+K, Ctrl+O
Точка останова
Показать следующую инструкцию Alt+Доп. кл. *
Выполнить до текущей позиции Ctrl+F10
Принудительное выполнение до текущей позиции
Задать следующую инструкцию Ctrl+Shift+F10
К дизассемблированному коду Ctrl+K, G
Добавить контрольное значение
Быстрая проверка... Shift+F9
Проблемы не найдены.
Глубина поиска: 3
0 Предупреждения

Задание

dumpbin /RAWDATA:BYTES /SECTION:.rdata CrackMe.exe > rdata.txt