API Deobfuscator: Resolving Obfuscated API Functions in Modern Packers

black hat USA 2015

2015.8.6.

Seokwoo Choi

UBM Tech

Outline

- Introduction
- API Deobfuscation Method
  - Memory Access Analysis for Dynamic Obfuscation
  - Iterative Run-until API method for Static Obfuscation
- Implementation
- Demo
- Conclusion

Why API deobfuscation matters?

- Malwares hide functionalities by API obfuscation
  - Commercial packers obfuscate API functions
  - Malware authors have their own API obfuscator
- No deobfuscation tools for some modern packers
  - x64 packers
  - Custom packers

API obfuscation techniques in modern packers

- Dynamic API Obfuscation
  - API functions are obfuscated during runtime
  - Instructions and addresses changes every run

```
helloworldmsgbox32_tiger_red.exe+0000100e call 0x68a0000
068a0000 mov edi, edi
068a0002 pushad
068a0003 pushad
068a0004 pushfd
068a0005 jmp 0x68a001a
068a001a xor ecx, 0x2bb296b6
068a0020 jmp 0x68a0031
....
068a032e pop eax
068a032f call 0x777de9ed
user32.dll+0005e9ed mov edi, edi
user32.dll+0005e9ef push ebp
```

Branch into a newly allocated block
during execution time
(obfuscated User32.dll :MessageBox)

API obfuscation techniques in modern packers

- Static API Obfuscation
  - API functions are obfuscated compile(packing) time
  - Instructions and addresses are the same

```
hw64_tmd233_tr.exe+0000000000001017 call 0x7ff6bbef63a0
hw64_tmd233_tr.exe+00000000002c63a0 sub rsp, 0x8
hw64_tmd233_tr.exe+00000000002c63a4 jmp 0x7ff6bbf00b79
hw64_tmd233_tr.exe+00000000002d0b79 push r10
hw64_tmd233_tr.exe+00000000002d0b7b push r10
hw64_tmd233_tr.exe+00000000002d0b7d add qword ptr [rsp], 0x72384261
hw64_tmd233_tr.exe+00000000002d0b85 jmp 0x7ff6bbf01d47
hw64_tmd233_tr.exe+00000000002d1d47 mov r10, qword ptr [rsp]
hw64_tmd233_tr.exe+00000000002d1d4b add rsp, 0x8
hw64_tmd233_tr.exe+00000000002d1d4f push rcx
hw64_tmd233_tr.exe+00000000002d1d50 jmp 0x7ff6bbf01efc
hw64_tmd233_tr.exe+00000000002d1efc mov rcx, 0x72384261
```

Branch into other section

……

```
hw64_tmd233_tr.exe+00000000002ccf29 push r13
hw64_tmd233_tr.exe+00000000002ccf2b mov r13, 0x8
hw64_tmd233_tr.exe+00000000002ccf32 sub rax, 0x5f6f805b
hw64_tmd233_tr.exe+00000000002ccf38 add rax, r13
hw64_tmd233_tr.exe+00000000002ccf3b jmp 0x7ff6bbf01830
hw64_tmd233_tr.exe+00000000002d1830 add rax, 0x5f6f805b
hw64_tmd233_tr.exe+00000000002d1836 pop r13
hw64_tmd233_tr.exe+00000000002d1838 xchg qword ptr [rsp], rax
hw64_tmd233_tr.exe+00000000002d183c jmp 0x7ff6bbf022a8
hw64_tmd233_tr.exe+00000000002d22a8 mov rsp, qword ptr [rsp]
hw64_tmd233_tr.exe+00000000002d22ac ret
```

API Call by ‘ret’ instruction

API Deobfuscation Goal

- After deobfuscation, we have
  - (Near) original entry point
  - Recovered API function calls at OEP
- With the deobfuscated image, we can do
  - Static analysis with disassembled and decompiled code
  - Dynamic analysis with debuggers

API Deobfuscation Methods

- How to deobfuscate API obfuscated binaries?
  - Dynamic API Obfuscation
    → Memory Access Analysis
  - Static API Obfuscation
    → Iterative Run-until-API Method
- How to evade anti-debugging?
  - Dynamic binary instrumentation (Intel Pin)
  - Anti-anti-debugger plugin in debuggers
  - Emulators

black hat

USA 2015

API Deobfuscation for Dynamic Obfuscators

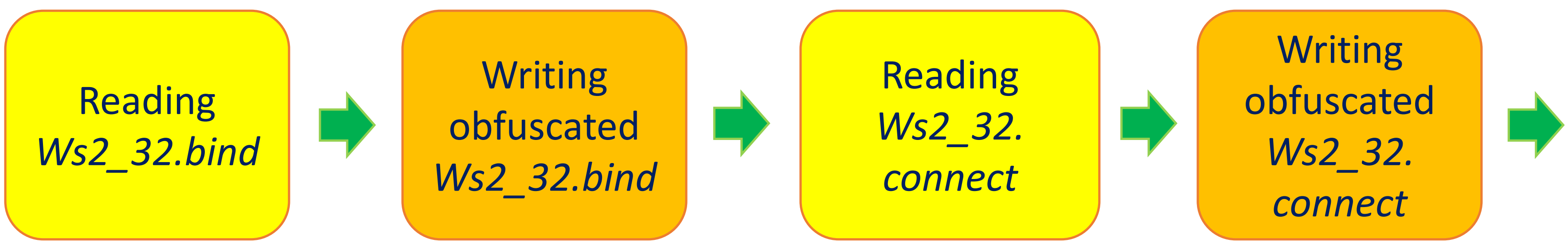
UBM Tech

API Deobfuscation for Dynamic Obfuscators

- Memory Access Analysis
  - Relate *memory reads on API function code* and corresponding *memory writes on obfuscated code*
    - Instruction addresses of obfuscated API function → Original API function
  - Recover original API function by the obfuscated call target address

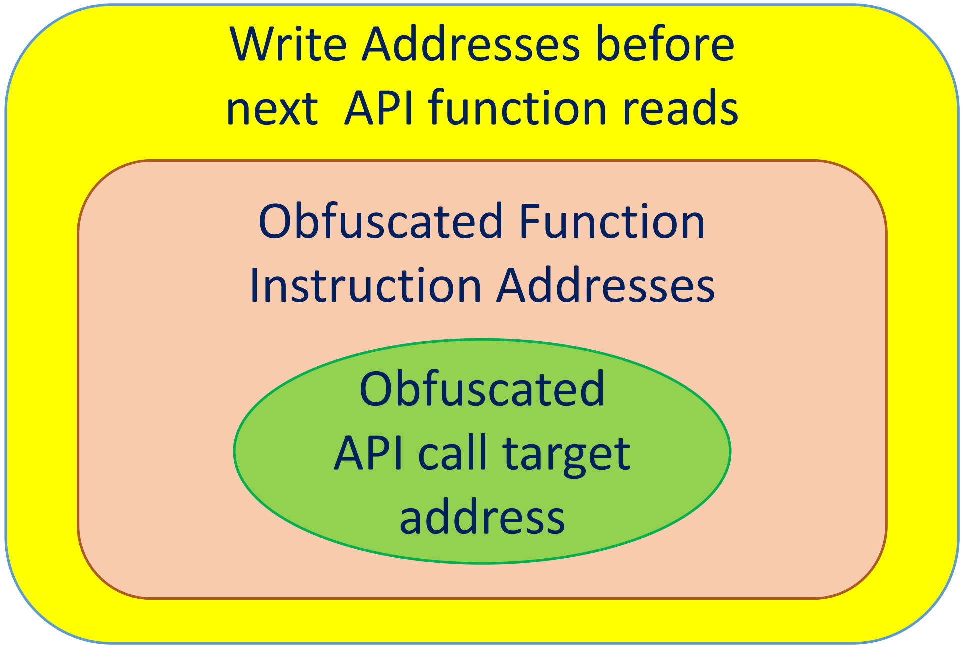
Dynamic Obfuscation Process

- What happens during runtime obfuscation?
  - Runtime obfuscator reads each function, obfuscates each instruction, writes the obfuscated code into a newly allocated memory
  - Each function is obfuscated in sequence



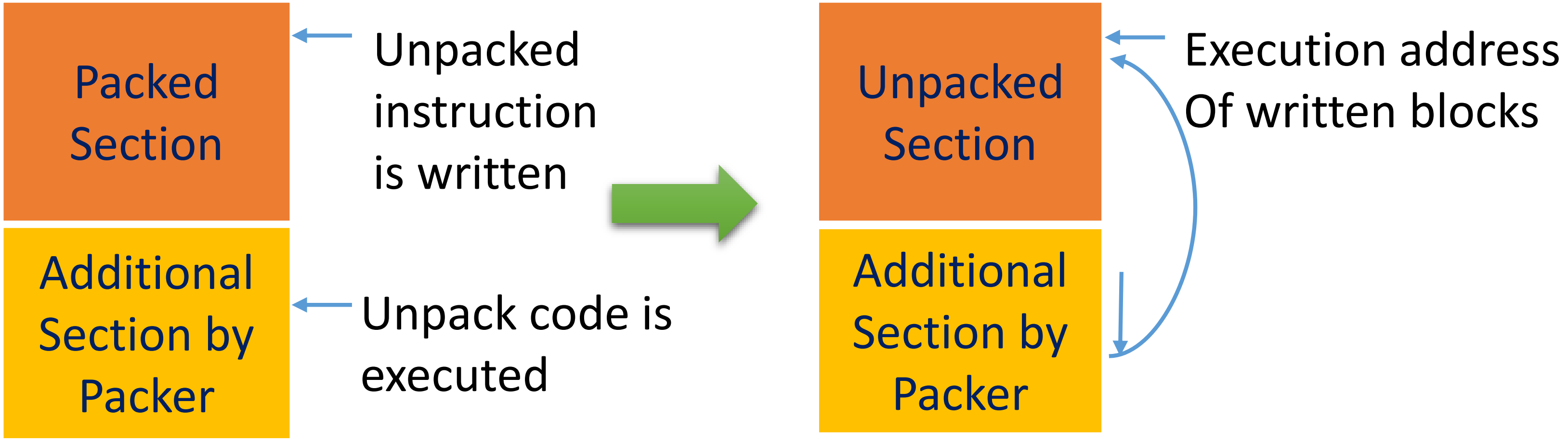
Memory Access Analysis

- How can we identify the original API function?
  - Record every memory write before the next API function or DLL reads
  - Limit the number of memory write for the last API function



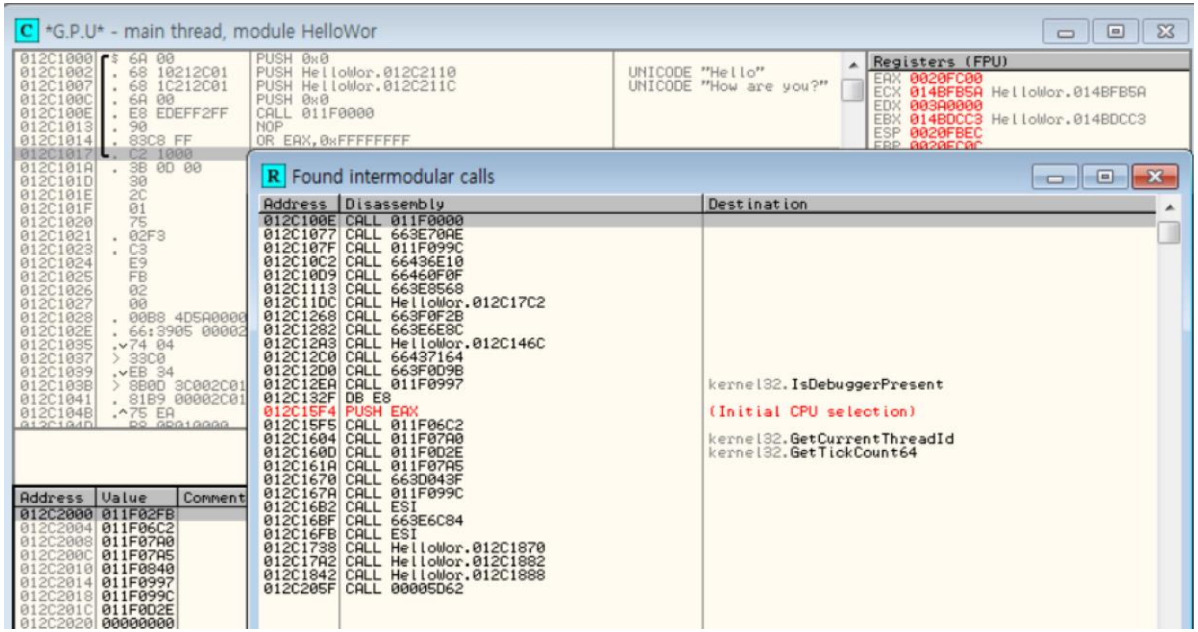
How to find OEP

- Find OEP
  - Record every memory write and execute
  - OEP is the Last written address that is executed
  - Check written memory blocks (1 block = 4 Kbytes) to save memory
  - OEP is in the original executable file sections



Obfuscated Call Identification

- Search for intermodular calls at OEP by pattern matching
  - Matched patterns may contain false positives
  - After target address resolution, misinterpreted instruction disappears



Obfuscated Call Resolution

- Direct call resolution
  - If the call targets are in the constructed map from obfuscated addresses to API function, modify call targets to the original API function address
  - Generate a text file that contains resolved API function calls and OEP

Obfuscated Call Resolution

- Indirect call resolution
  - Original segments (.text, .idata, …) are merged into one segment by packing
  - Identify a memory block that contains successive obfuscated API function addresses
  - Modify obfuscated call addresses in the IAT candidate with the original API function

Obfuscated Call Resolution

- Example: API Deobufscation Information

OEP:0000112d

00002000	addr	ntdll.dll	RtlDecodePointer
00002004	addr	kernel32.dll	GetSystemTimeAsFileTime
00002008	addr	kernel32.dll	GetCurrentThreadId
0000200c	addr	kernel32.dll	QueryPerformanceCounter
00002010	addr	kernel32.dll	IsProcessorFeaturePresent
00002014	addr	kernel32.dll	IsDebuggerPresent
00002018	addr	ntdll.dll	RtlEncodePointer
0000201c	addr	kernel32.dll	GetTickCount64
0000203c	addr	ntdll.dll	RtlFreeHeap
0000209c	addr	user32.dll	MessageBoxW
0000100e	call	user32.dll	MessageBoxW
0000107f	call	ntdll.dll	RtlEncodePointer
000012ea	call	kernel32.dll	IsDebuggerPresent
000015f5	call	kernel32.dll	GetSystemTimeAsFileTime
00001604	call	kernel32.dll	GetCurrentThreadId
0000160d	call	kernel32.dll	GetTickCount64
0000161a	call	kernel32.dll	QueryPerformanceCounter
0000167a	call	ntdll.dll	RtlEncodePointer

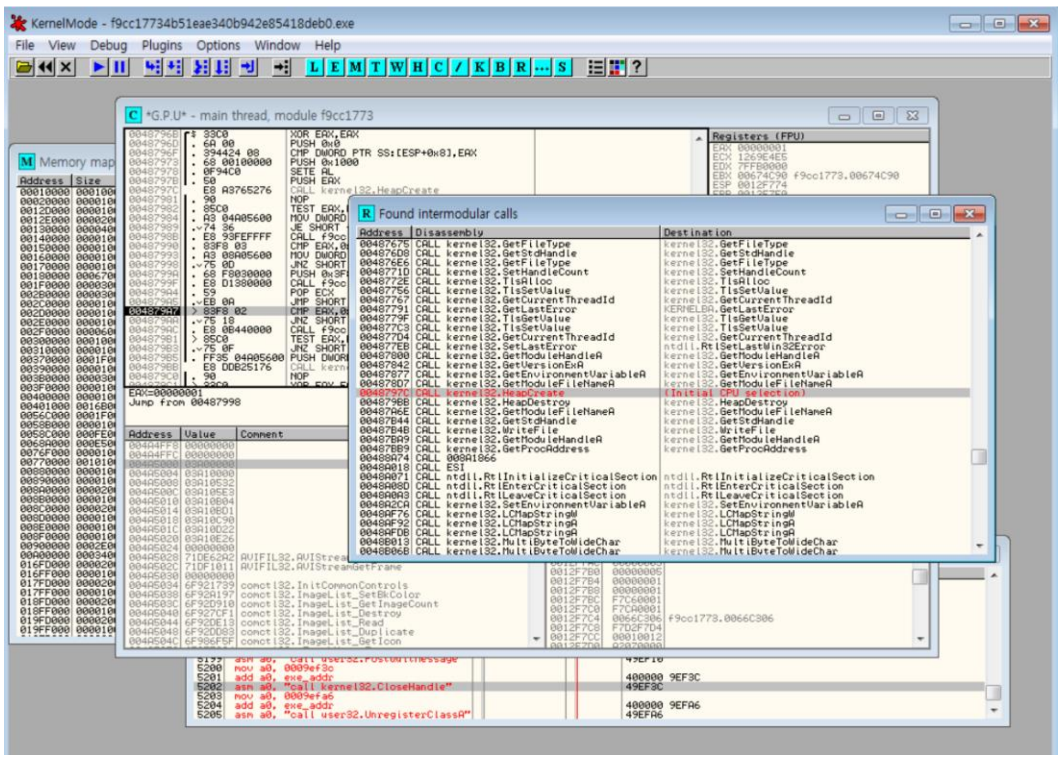
Addresses are in RVA

Debugging Deobfuscated Binary

- Generating a debugger script to resolve API calls
  - The text file generated by the memory access analyzer contains OEP, resolved obfuscated addresses
  - Implemented a python script to generate a debugger script that execute until OEP and resolve obfuscated addresses

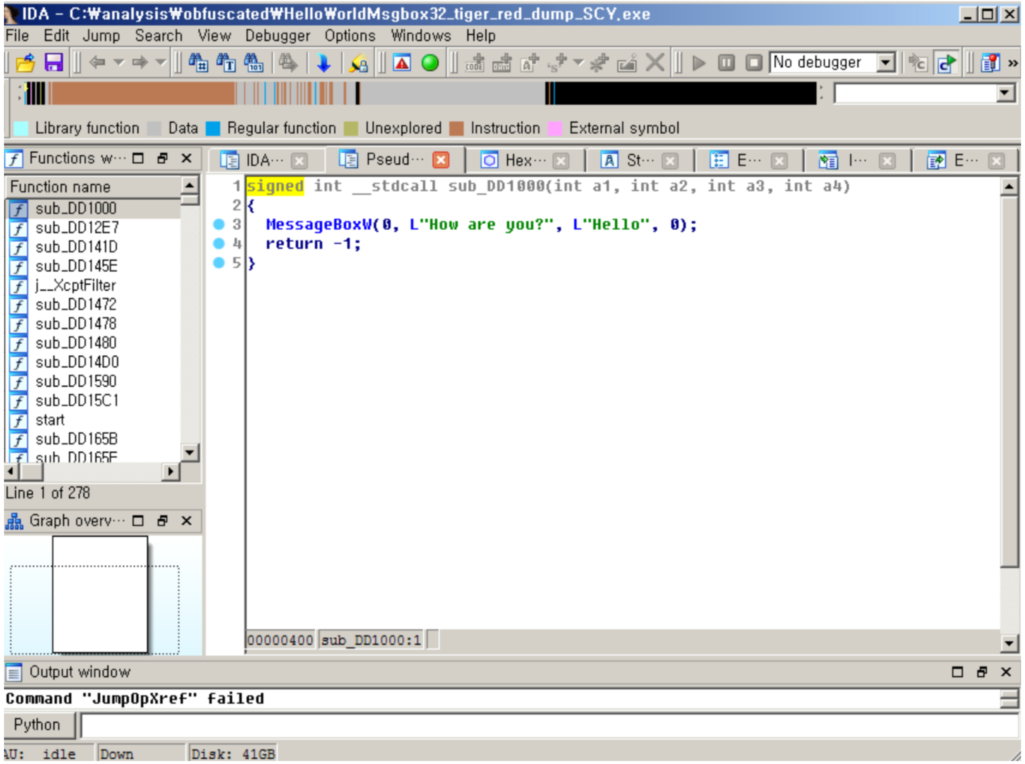
Reversing with API Deobfuscator

- Debugging x86 binary with Ollydbg after running deobfuscation script



Reversing with API Deobfuscator

- Decompiled code with dumped file



black hat USA 2015

API Deobfuscation for Static Obfuscators

API Deobfuscation for Static Obfuscators

- Static obfuscation pattern at OEP
  - Obfuscated call pattern
    - “Call qword ptr [___]” is changed into “Call rel32” when obfuscated
  - Obfuscated call run into API function
    - Stack shape is preserved
    - API call instruction and the first few instructions in the API function are obfuscated
    - After executing obufscated instructions, execution reaches an instruction in the original API function

Obfuscated Call Identification

- Search obfuscated call by pattern
  - CALL rel32 – is a candidate
  - Check whether the address is in another section of the process

```
1000   48 83 EC 28             sub rsp,28
1004   4C 8D 05 BD 11 00 00    lea r8,qword ptr ds:[7FF6BBC321C8]
100B   48 8D 15 C6 11 00 00    lea rdx,qword ptr ds:[7FF6BBC321D8]
1012   45 33 C9                xor r9d,r9d
1015   33 C9                   xor ecx,ecx
1017   E8 84 53 2C 00          call hw64_tmd233_tr.7FF6BBEF63A0
101C   00 83 C8 FF 48 83       add byte ptr ds:[rbx-7CB70038],al
1022   C4                      ???
1023   28 C3                   sub bl,al
```

```
10B9   E8 EF 11 2D 00          call hw64_tmd233_tr.7FF6BBF022AD
10BE   00 48 83                add byte ptr ds:[rax-7D],cl
10C1   C9                      leave
10C2   FF E8                   jmp far EA:0
10C4   01 B7 2C 00 00 48       add dword ptr ds:[rdi+4800002C],esi
```

……

Call rel32; db 00
'00' after call break alignment
so thataA few incorrect disassembled code occur

Obfuscated Call Resoultion

- Obfuscated code is executed until API function
- Run-until-API method
  - Change RIP into candidate API call address
  - Run until API function

```
hw64_tmd233_tr.exe+0000000000001017 call 0x7ff6bbef63a0
hw64_tmd233_tr.exe+00000000002c63a0 sub rsp, 0x8
hw64_tmd233_tr.exe+00000000002c63a4 jmp 0x7ff6bbf00b79
hw64_tmd233_tr.exe+00000000002d0b79 push r10
hw64_tmd233_tr.exe+00000000002d0b7b push r10
hw64_tmd233_tr.exe+00000000002d0b7d add qword ptr [rsp], 0x72384261
hw64_tmd233_tr.exe+00000000002d0b85 jmp 0x7ff6bbf01d47
hw64_tmd233_tr.exe+00000000002d1d47 mov r10, qword ptr [rsp]
hw64_tmd233_tr.exe+00000000002d1d4b add rsp, 0x8
hw64_tmd233_tr.exe+00000000002d1d4f push rcx
hw64_tmd233_tr.exe+00000000002d1d50 jmp 0x7ff6bbf01efc
hw64_tmd233_tr.exe+00000000002d1efc mov rcx, 0x72384261
```

Obfuscated Call Start

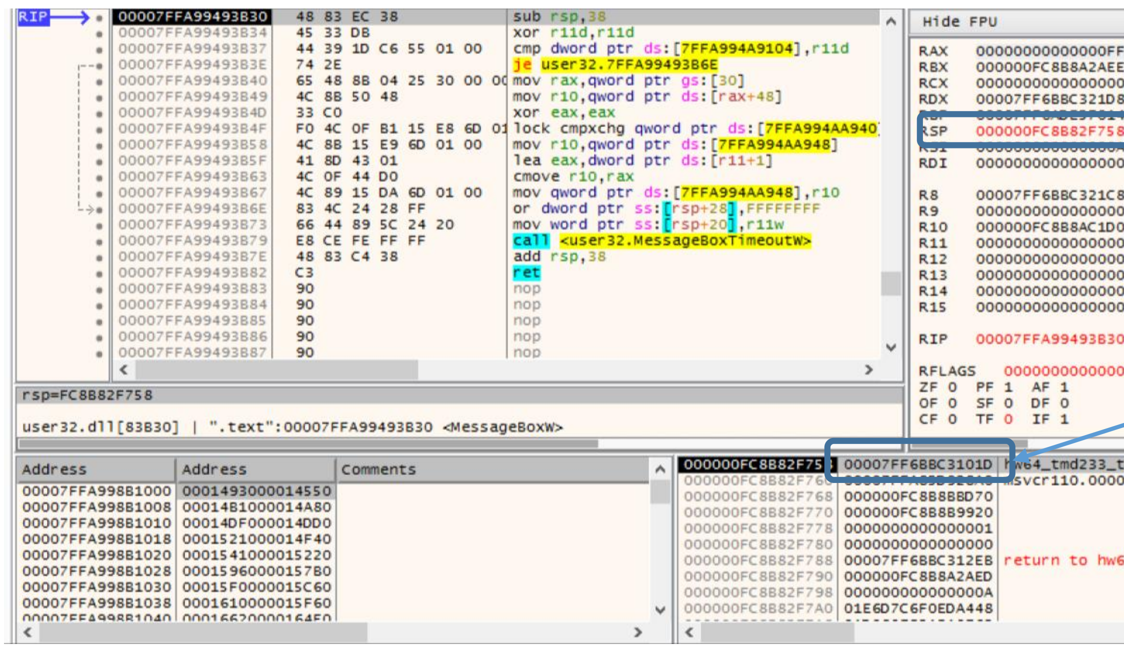
……

```
hw64_tmd233_tr.exe+00000000002ccf3b jmp 0x7ff6bbf01830
hw64_tmd233_tr.exe+00000000002d1830 add rax, 0x5f6f805b
hw64_tmd233_tr.exe+00000000002d1836 pop r13
hw64_tmd233_tr.exe+00000000002d1838 xchg qword ptr [rsp], rax
hw64_tmd233_tr.exe+00000000002d183c jmp 0x7ff6bbf022a8
hw64_tmd233_tr.exe+00000000002d22a8 mov rsp, qword ptr [rsp]
hw64_tmd233_tr.exe+00000000002d22ac ret
user32.dll+0000000000083b30 sub rsp, 0x38
user32.dll+0000000000083b34 xor r11d, r11d
user32.dll+0000000000083b37 cmp dword ptr [rip+0x155c6], r11d
```

Execute until API address is met

Obfuscated Call Resolution

- Integrity check
  - We need to check whether the stack pointer and the stack content is preserved after executing obfuscate call



Check
Stack Pointer

Check Stack &
Return Address

Iterative run-until-API Method

- Apply run-until API method repeatedly on candidate obfuscated calls
  - Save context & Restore

```
0000000000001017  call  user32.dll  MessageBoxW
00000000000010b9  call  msvcr110.dll  __set_app_type
00000000000010c3  call  ntdll.dll RtlEncodePointer
000000000000110f  call  msvcr110.dll  __setusermatherr
0000000000001121  call  msvcr110.dll  _configthreadlocale
0000000000001174  call  msvcr110.dll  __getmainargs
00000000000012fc  call  msvcr110.dll  exit
000000000000130b  call  msvcr110.dll  _cexit
000000000000132e  call  msvcr110.dll  _ismbblead
000000000000135e  call  msvcr110.dll  _exit
000000000000136e  call  msvcr110.dll  _cexit
0000000000001395  call  kernel32.dll  IsDebuggerPresent
00000000000014fc  goto  msvcr110.dll  __XcptFilter
```

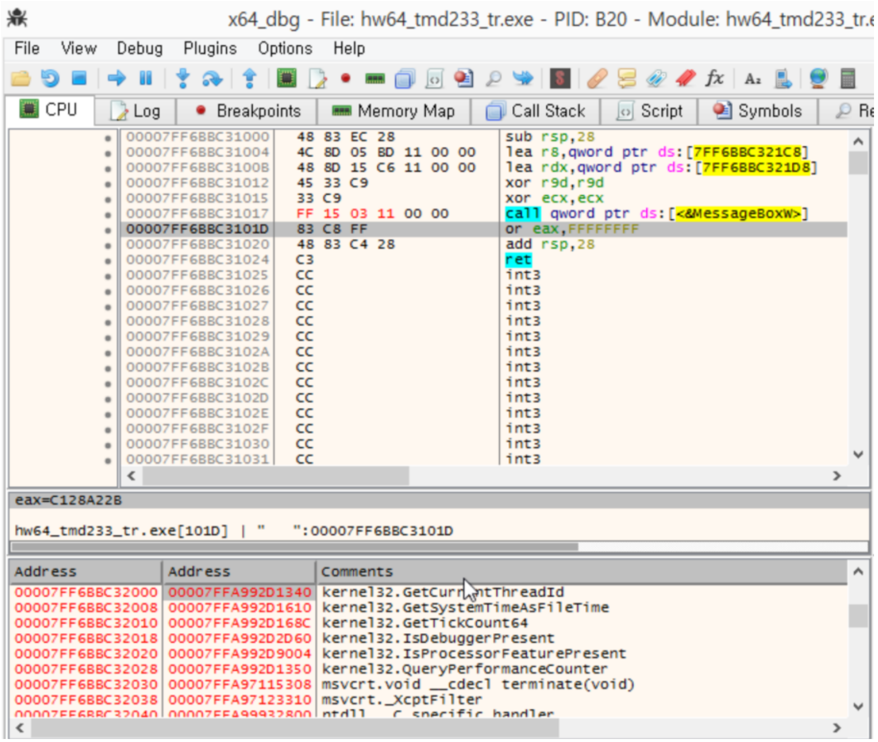
....

Iterative run-until-API Method

- Iterative run-until-API method can be applied to various packers
  - VMP: API function call is virtualization-obfuscated
  - Themida64: API function call is mutated
  - Obsidium: The first few instructions in an API function are obfuscated
  - Custom packers
  - But, at last, execution is redirected into a real API function

Reversing with API Deobfuscator

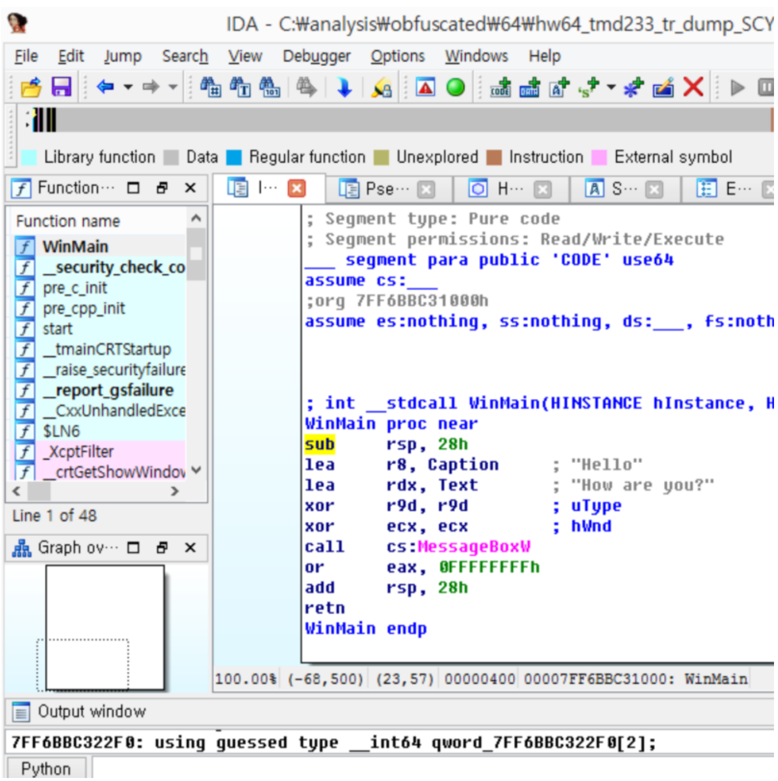
- Debugging x64 binary with x64DBG after deobfuscation



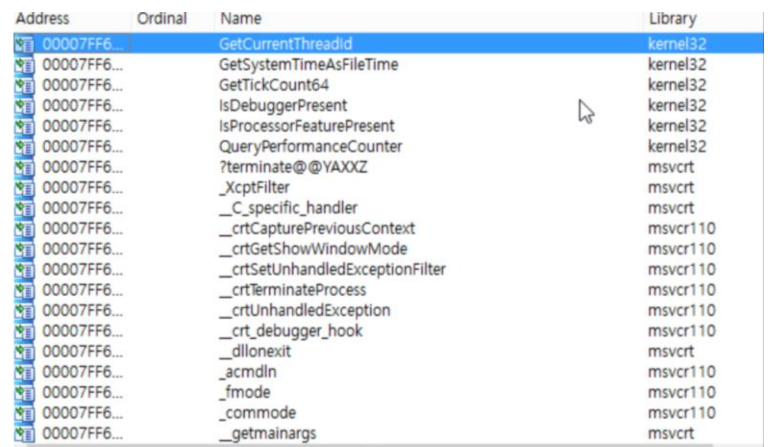
Run correctly with resolved API call

Reversing with API Deobfuscator

- Dumping x86/64 binary and static analysis with IDA Pro



API call recovered



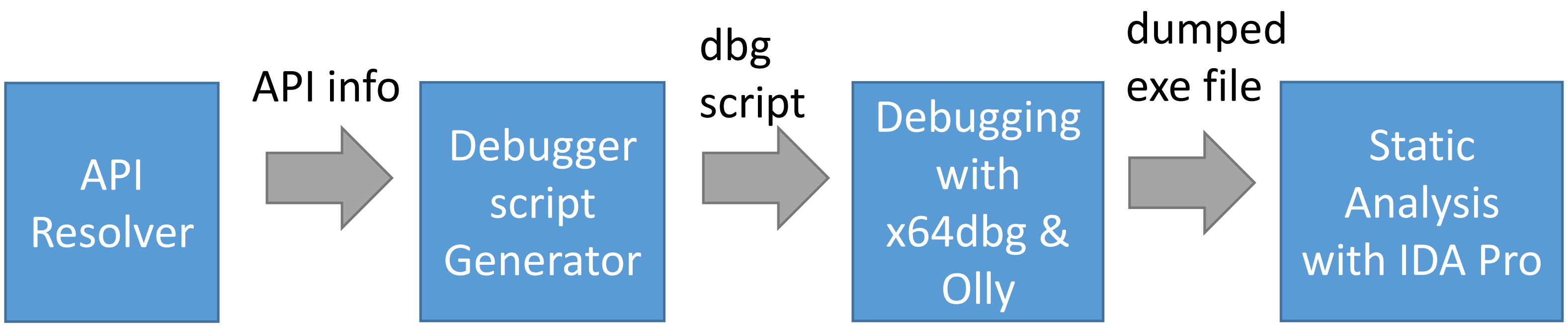
IAT recovered

Implementation

Implementation Detail

- Pin tool to resolve API Address
  - Windows 8.1/7 – 32/64 bit (on VMWare)
  - Visual Studio 2013
  - Intel Pin 2.14
- Python script to patch obfuscated call
- Reversing tools
  - X64dbg
  - IDA

Deobfuscation Process



Demo

Reversing Packed Binary with API Deobfuscator

- Packed 32/64 bit samples
- Commercial packer packed 32bit malware

Conclusion

Conclusion

- Suggested two methods for API deobfuscatoin
  - Memory access analysis for dynamic obfuscation
  - Run-until-API method for static obfuscation
- Commercial packer protected binary can be analyzed using API deobfuscator
  - Using debugger
  - Using disassembler & decompiler

Limitation

- Depending on DBI tools
  - Packers can detect DBI tools
    - Defeating the transparency feature of DBI (BH US'14)
    - Ex) Obsidium detect Intel Pin as a debugger
  - DBI tools crash in some applications
- Static whole function obfuscated code cannot be deobfuscated
  - No instructions in the original API function is executed when the whole function is obfuscated

Future Work

- Anti-anti-debugging
  - Building x86/64 emulator for unpacking

- API function resolution
  - Code optimization and binary diffing for static whole function obfuscation
  - Backward dependence analysis for custom packers