blackhat ASIA 2015

API Deobfuscator: Identifying Runtime-obfuscated API calls via Memory Access Analysis

2015. 3. 26.
Seokwoo Choi

Overview

- Runtime API obfuscation
- Memory access analysis
- Identifying original API functions
- Patching obfuscated API calls
- Analyzing deobufscated binary
- Related work
- Conclusion

Run-time API Obfuscation

- Code obfuscation is applied on
  - Source code
  - Object file
  - Executable file
  - In-memory executable file image

Compile-time obfuscation

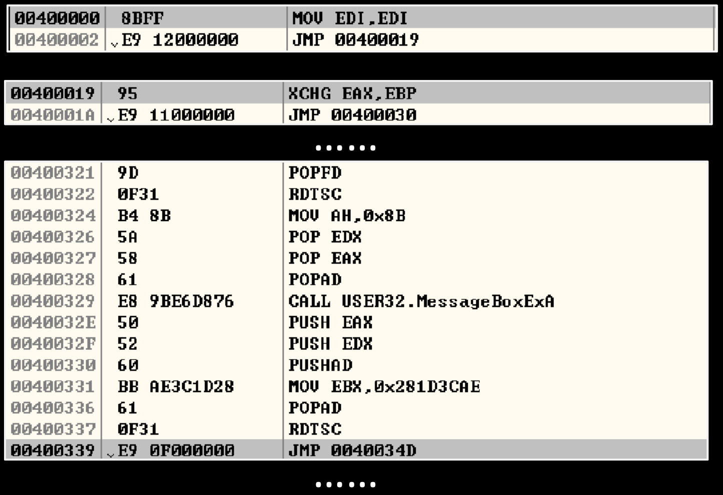
Run-time obfuscation

Run-time API Obfuscation

- Runtime code obfuscation techniques embed obfuscation engine in executable file and apply code obfuscation techniques on memory loaded executable file image
- Types of obfuscating transformations are selected randomly so that obfuscated binary image is different each time a packed file executes

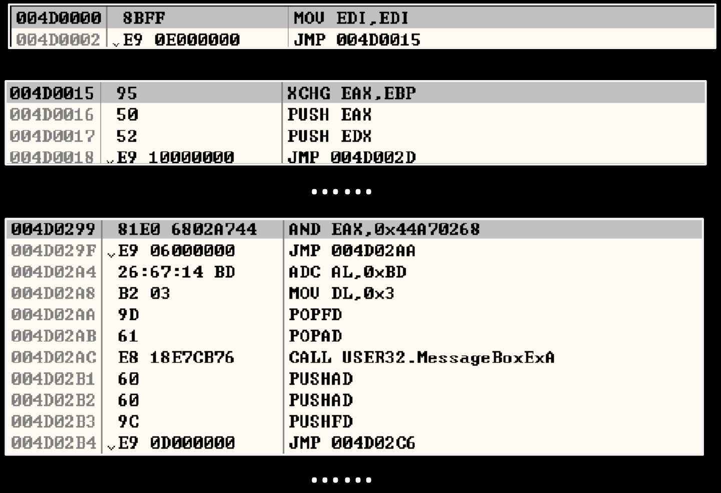
Run-time API Obfuscation

- Call addresses and obfuscated function code is changing for each execution



```
00400000  8BFF          MOV EDI,EDI
00400002  E9 12000000   JMP 00400019

00400019  95            XCHG EAX,EBP
0040001A  E9 11000000   JMP 00400030
......
00400321  9D            POPFD
00400322  0F31          RDTSC
00400324  B4 8B         MOV AH,0x8B
00400326  5A            POP EDX
00400327  58            POP EAX
00400328  61            POPAD
00400329  E8 9BE6D876   CALL USER32.MessageBoxExA
0040032E  50            PUSH EAX
0040032F  52            PUSH EDX
00400330  60            PUSHAD
00400331  BB AE3C1D28   MOV EBX,0x281D3CAE
00400336  61            POPAD
00400337  0F31          RDTSC
00400339  E9 0F000000   JMP 0040034D
......
```

1st time user32.MessageBoxA is obfuscated

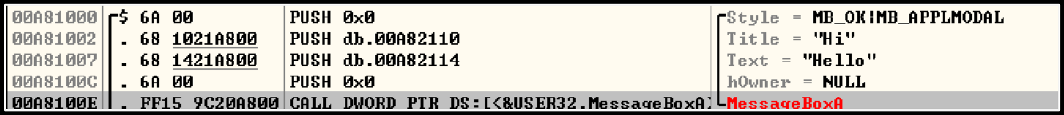


```
004D0000  8BFF            MOV EDI,EDI
004D0002  E9 0E000000     JMP 004D0015

004D0015  95              XCHG EAX,EBP
004D0016  50              PUSH EAX
004D0017  52              PUSH EDX
004D0018  E9 10000000     JMP 004D002D
......
004D0299  81E0 6802A744   AND EAX,0x44A70268
004D029F  E9 06000000     JMP 004D02AA
004D02A4  26:67:14 BD     ADC AL,0xBD
004D02A8  B2 03           MOV DL,0x3
004D02AA  9D              POPFD
004D02AB  61              POPAD
004D02AC  E8 18E7CB76     CALL USER32.MessageBoxExA
004D02B1  60              PUSHAD
004D02B2  60              PUSHAD
004D02B3  9C              PUSHFD
004D02B4  E9 0D000000     JMP 004D02C6
......
```

2nd time user32.MessageBoxA is obfuscated

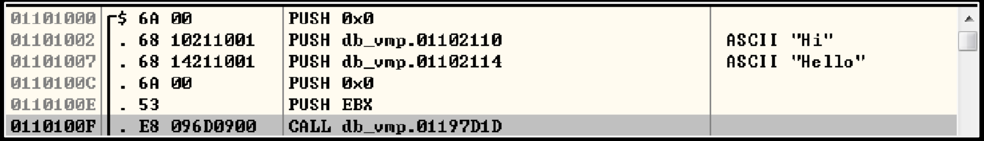
API Obfuscation Example

- Without runtime API obfuscation, setting breakpoint on API function works

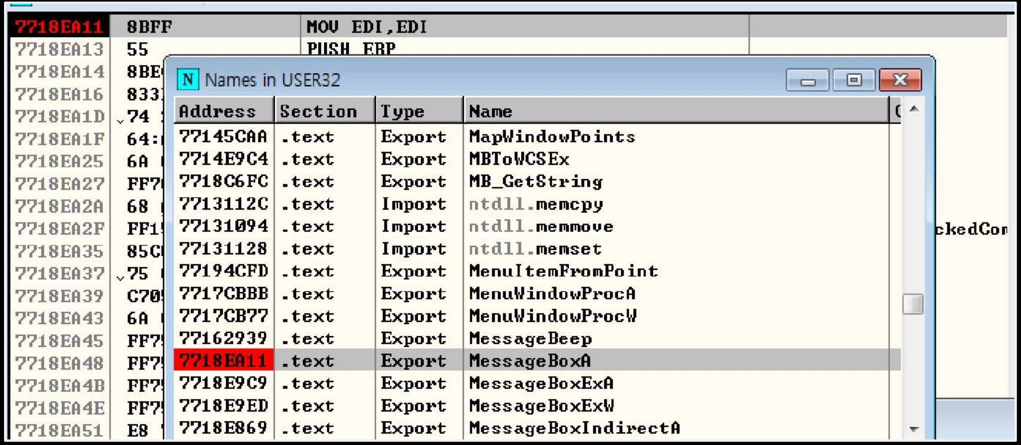


```
00A81000 ┌$ 6A 00          PUSH 0x0                                  ┌Style = MB_OK|MB_APPLMODAL
00A81002 │. 68 1021A800    PUSH db.00A82110                          │Title = "Hi"
00A81007 │. 68 1421A800    PUSH db.00A82114                          │Text = "Hello"
00A8100C │. 6A 00          PUSH 0x0                                  │hOwner = NULL
00A8100E │. FF15 9C20A800  CALL DWORD PTR DS:[<&USER32.MessageBoxA]  └MessageBoxA
```

After VMP Packing



```
01101000 ┌$ 6A 00          PUSH 0x0
01101002 │. 68 10211001    PUSH db_vmp.01102110      ASCII "Hi"
01101007 │. 68 14211001    PUSH db_vmp.01102114      ASCII "Hello"
0110100C │. 6A 00          PUSH 0x0
0110100E │. 53             PUSH EBX
0110100F │. E8 096D0900    CALL db_vmp.01197D1D
```



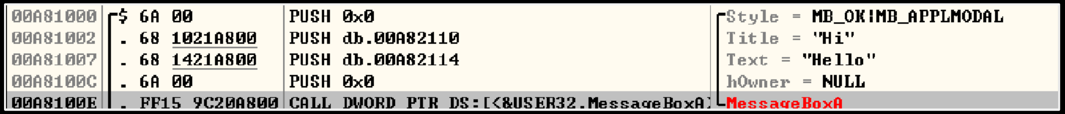
```
7718EA11   8BFF     MOV EDI,EDI
7718EA13   55       PUSH EBP
7718EA14   8BE
7718EA16   833
7718EA1D ⌄ 74
7718EA1F   64:
7718EA25   6A
7718EA27   FF7
7718EA2A   68
7718EA2F   FF1                                  ckedCor
7718EA35   85C
7718EA37 ⌄ 75
7718EA39   C70
7718EA43   6A
7718EA45   FF7
7718EA48   FF7
7718EA4B   FF7
7718EA4E   FF7
7718EA51   E8
```

Address	Section	Type	Name
77145CAA	.text	Export	MapWindowPoints
7714E9C4	.text	Export	MBToWCSEx
7718C6FC	.text	Export	MB_GetString
7713112C	.text	Import	ntdll.memcpy
77131094	.text	Import	ntdll.memmove
77131128	.text	Import	ntdll.memset
77194CFD	.text	Export	MenuItemFromPoint
7717CBBB	.text	Export	MenuWindowProcA
7717CB77	.text	Export	MenuWindowProcW
77162939	.text	Export	MessageBeep
7718EA11	.text	Export	MessageBoxA
7718E9C9	.text	Export	MessageBoxExA
7718E9ED	.text	Export	MessageBoxExW
7718E869	.text	Export	MessageBoxIndirectA

No change in API function code

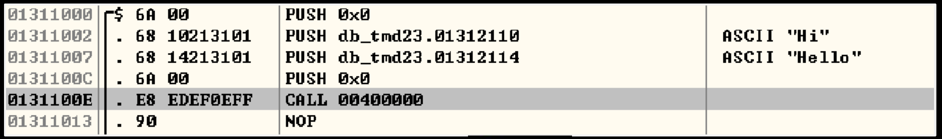
API Obfuscation Example

- With runtime obfuscation, API function is obfuscated and hidden



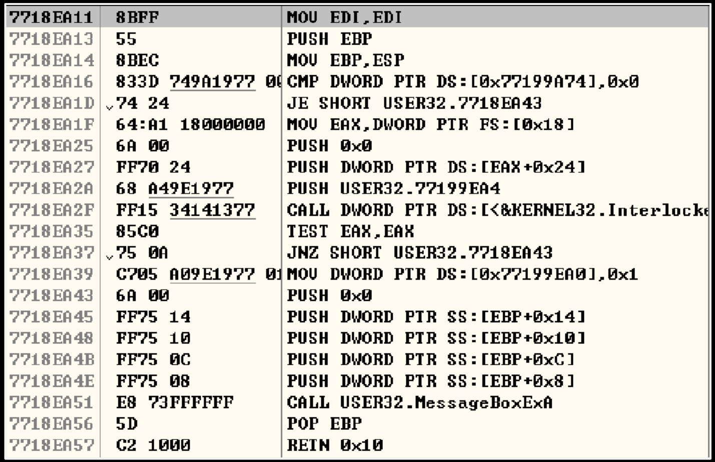
```
00A81000 ┌$ 6A 00          PUSH 0x0                                   ┌Style = MB_OK|MB_APPLMODAL
00A81002 │. 68 1021A800    PUSH db.00A82110                           │Title = "Hi"
00A81007 │. 68 1421A800    PUSH db.00A82114                           │Text = "Hello"
00A8100C │. 6A 00          PUSH 0x0                                   │hOwner = NULL
00A8100E │. FF15 9C20A800  CALL DWORD PTR DS:[<&USER32.MessageBoxA]   └MessageBoxA
```

After Themida Packing



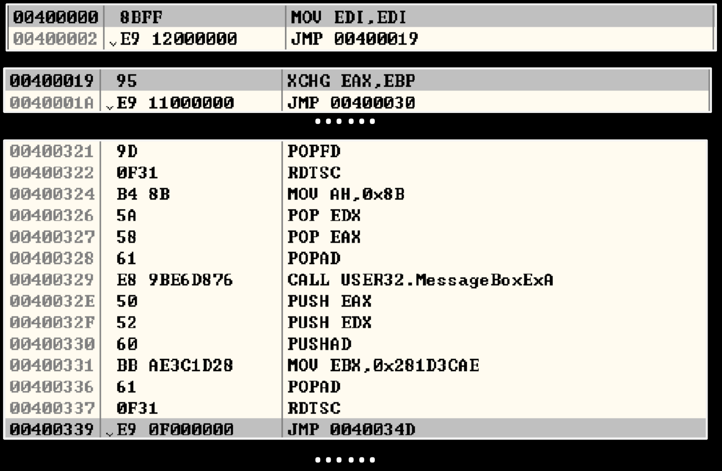
```
01311000 ┌$ 6A 00          PUSH 0x0
01311002 │. 68 10213101    PUSH db_tmd23.01312110        ASCII "Hi"
01311007 │. 68 14213101    PUSH db_tmd23.01312114        ASCII "Hello"
0131100C │. 6A 00          PUSH 0x0
0131100E │. E8 EDEF0EFF    CALL 00400000
01311013 │. 90             NOP
```

Calling Obfuscated API function



```
7718EA11   8BFF              MOV EDI,EDI
7718EA13   55                PUSH EBP
7718EA14   8BEC              MOV EBP,ESP
7718EA16   833D 749A1977 00  CMP DWORD PTR DS:[0x77199A74],0x0
7718EA1D  ˬ74 24             JE SHORT USER32.7718EA43
7718EA1F   64:A1 18000000    MOV EAX,DWORD PTR FS:[0x18]
7718EA25   6A 00             PUSH 0x0
7718EA27   FF70 24           PUSH DWORD PTR DS:[EAX+0x24]
7718EA2A   68 A49E1977       PUSH USER32.77199EA4
7718EA2F   FF15 34141377     CALL DWORD PTR DS:[<&KERNEL32.Interlocke
7718EA35   85C0              TEST EAX,EAX
7718EA37  ˬ75 0A             JNZ SHORT USER32.7718EA43
7718EA39   C705 A09E1977 01  MOV DWORD PTR DS:[0x77199EA0],0x1
7718EA43   6A 00             PUSH 0x0
7718EA45   FF75 14           PUSH DWORD PTR SS:[EBP+0x14]
7718EA48   FF75 10           PUSH DWORD PTR SS:[EBP+0x10]
7718EA4B   FF75 0C           PUSH DWORD PTR SS:[EBP+0xC]
7718EA4E   FF75 08           PUSH DWORD PTR SS:[EBP+0x8]
7718EA51   E8 73FFFFFF       CALL USER32.MessageBoxExA
7718EA56   5D                POP EBP
7718EA57   C2 1000           RETN 0x10
```

Original user32.MessageBoxA



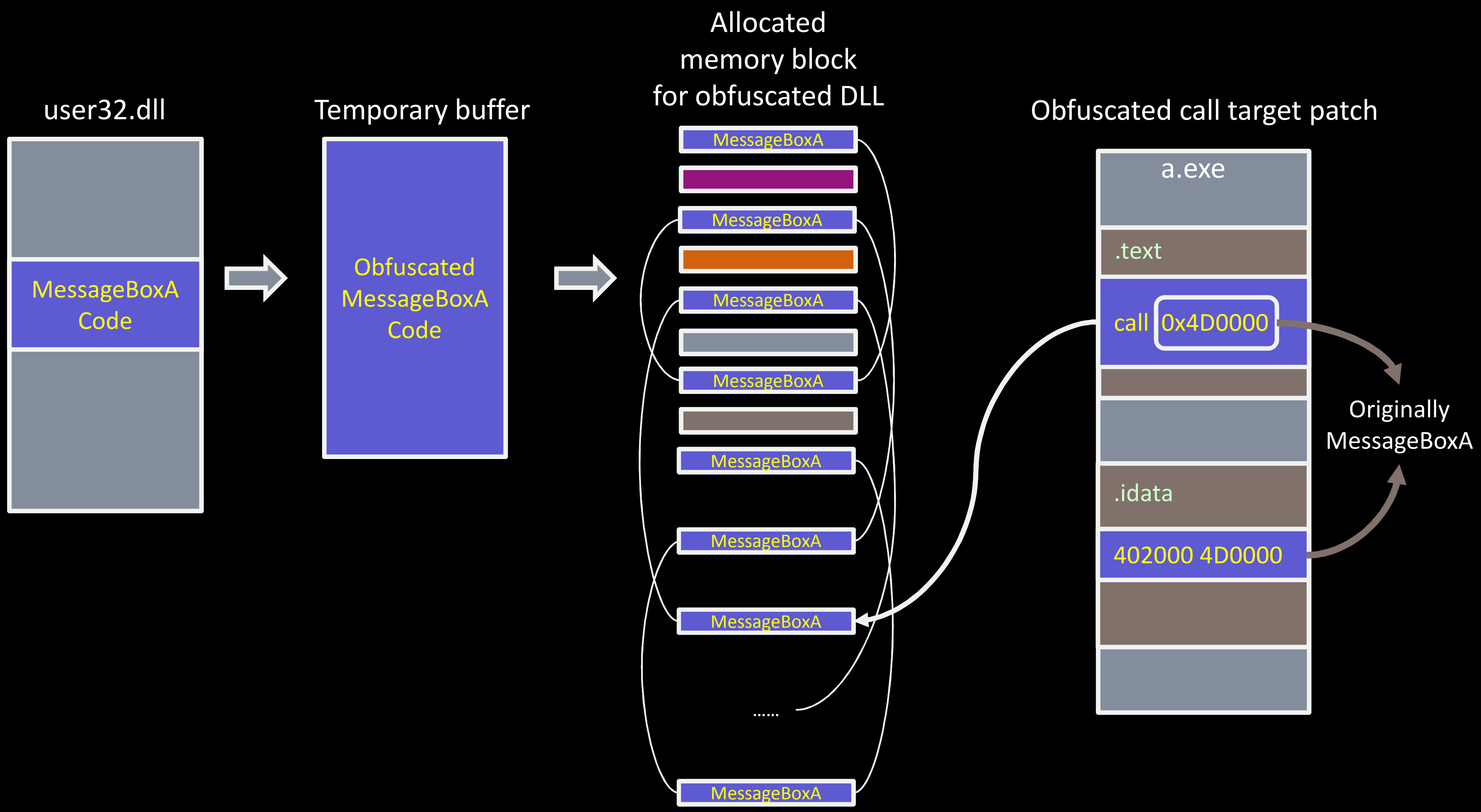
```
00400000   8BFF           MOV EDI,EDI
00400002  ˬE9 12000000    JMP 00400019

00400019   95             XCHG EAX,EBP
0040001A  ˬE9 11000000    JMP 00400030
......
00400321   9D             POPFD
00400322   0F31           RDTSC
00400324   B4 8B          MOV AH,0x8B
00400326   5A             POP EDX
00400327   58             POP EAX
00400328   61             POPAD
00400329   E8 9BE6D876    CALL USER32.MessageBoxExA
0040032E   50             PUSH EAX
0040032F   52             PUSH EDX
00400330   60             PUSHAD
00400331   BB AE3C1D28    MOV EBX,0x281D3CAE
00400336   61             POPAD
00400337   0F31           RDTSC
00400339  ˬE9 0F000000    JMP 0040034D
......
```

user32.MessageBoxA is obfuscated

How to deobfuscate API calls?

- Observation
  - Each function is obfuscated in sequence
  - For each API function, every instruction is read and obfuscated instructions are written

Observation: Obfuscation Process

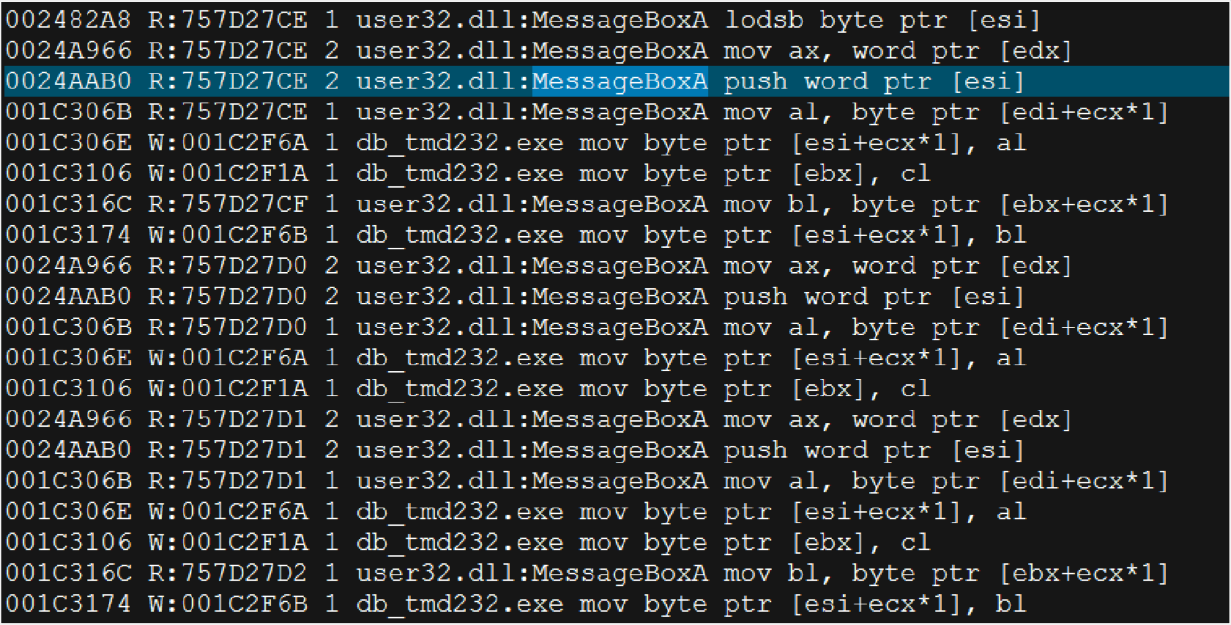


Identifying Original API Function

- Idea
  - — Relate *memory reads on API function code* and corresponding *memory writes on obfuscated code*
    - (Original API function address ← Addresses of obfuscated API function)
  - — Recover original API function by the obfuscated call target address

Memory Access Analysis

- API function memory reads are clustered
  - Memory reads occurs every byte in an original API function code

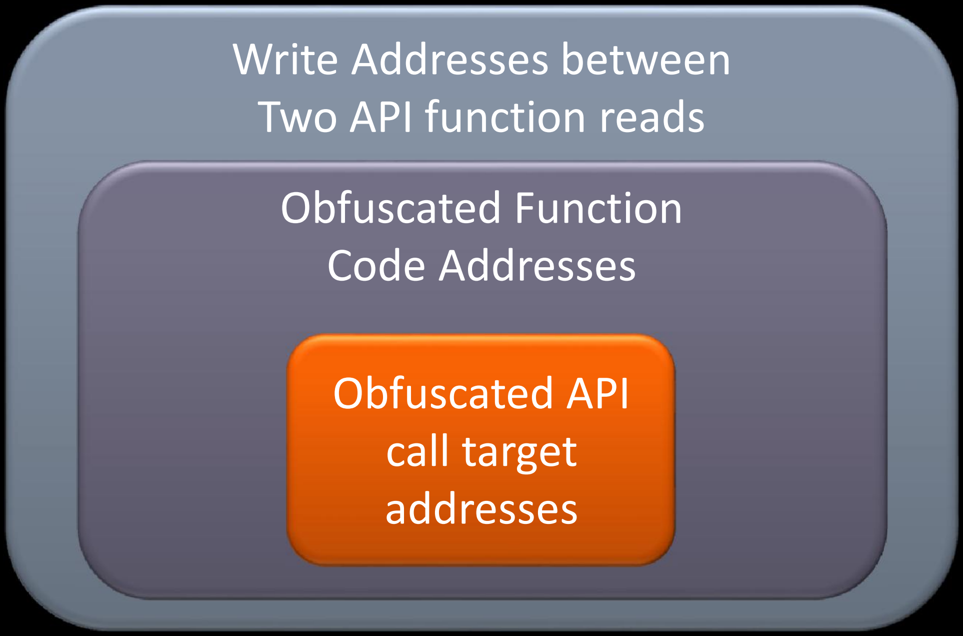


```
002482A8 R:757D27CE 1 user32.dll:MessageBoxA lodsb byte ptr [esi]
0024A966 R:757D27CE 2 user32.dll:MessageBoxA mov ax, word ptr [edx]
0024AAB0 R:757D27CE 2 user32.dll:MessageBoxA push word ptr [esi]
001C306B R:757D27CE 1 user32.dll:MessageBoxA mov al, byte ptr [edi+ecx*1]
001C306E W:001C2F6A 1 db_tmd232.exe mov byte ptr [esi+ecx*1], al
001C3106 W:001C2F1A 1 db_tmd232.exe mov byte ptr [ebx], cl
001C316C R:757D27CF 1 user32.dll:MessageBoxA mov bl, byte ptr [ebx+ecx*1]
001C3174 W:001C2F6B 1 db_tmd232.exe mov byte ptr [esi+ecx*1], bl
0024A966 R:757D27D0 2 user32.dll:MessageBoxA mov ax, word ptr [edx]
0024AAB0 R:757D27D0 2 user32.dll:MessageBoxA push word ptr [esi]
001C306B R:757D27D0 1 user32.dll:MessageBoxA mov al, byte ptr [edi+ecx*1]
001C306E W:001C2F6A 1 db_tmd232.exe mov byte ptr [esi+ecx*1], al
001C3106 W:001C2F1A 1 db_tmd232.exe mov byte ptr [ebx], cl
0024A966 R:757D27D1 2 user32.dll:MessageBoxA mov ax, word ptr [edx]
0024AAB0 R:757D27D1 2 user32.dll:MessageBoxA push word ptr [esi]
001C306B R:757D27D1 1 user32.dll:MessageBoxA mov al, byte ptr [edi+ecx*1]
001C306E W:001C2F6A 1 db_tmd232.exe mov byte ptr [esi+ecx*1], al
001C3106 W:001C2F1A 1 db_tmd232.exe mov byte ptr [ebx], cl
001C316C R:757D27D2 1 user32.dll:MessageBoxA mov bl, byte ptr [ebx+ecx*1]
001C3174 W:001C2F6B 1 db_tmd232.exe mov byte ptr [esi+ecx*1], bl
```

Memory R/W Traces

Memory Access Analysis

- Approximate API function memory writes
  - Record every memory write before the next API function or DLL reads
  - Limit the number of memory write



Write Addresses between Two API function reads

Obfuscated Function Code Addresses

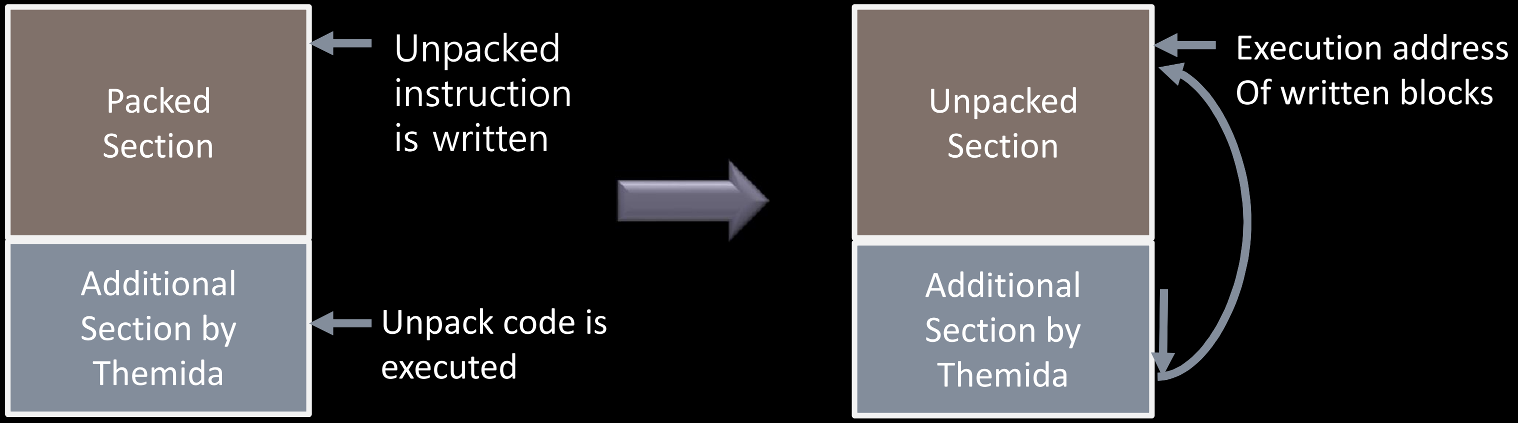
Obfuscated API call target addresses

Building Memory Access Analyzer

- Implemented as a Pin tool
  - Records memory reads on API functions
  - Records memory writes on newly allocated memory block
  - Construct a map from each API function to memory write addresses (a superset of obfuscated code addresses)
  - Pause at OEP

Building Memory Access Analyzer

- If an address in written memory block is executed, the address is a candidate of OEP
  - Check written memory blocks (1 block = 4 Kbytes) to save memory
  - OEP is in the original executable file sections


Packed Section
Additional Section by Themida
Unpacked instruction is written
Unpack code is executed
Unpacked Section
Additional Section by Themida
Execution address Of written blocks

Identifying Obfuscated API Call

- Identifying obfuscated calls that use direct addresses
  - At OEP, search for all external call (to another memory segments) from original executable section
  - Pattern matching is used to identify external calls
    - Matched patterns may contain misinterpreted bytes
    - After target address resolution, misinterpreted instruction disappears

Identifying Obfuscated API Call

- 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

Identifying Obfuscated API Call

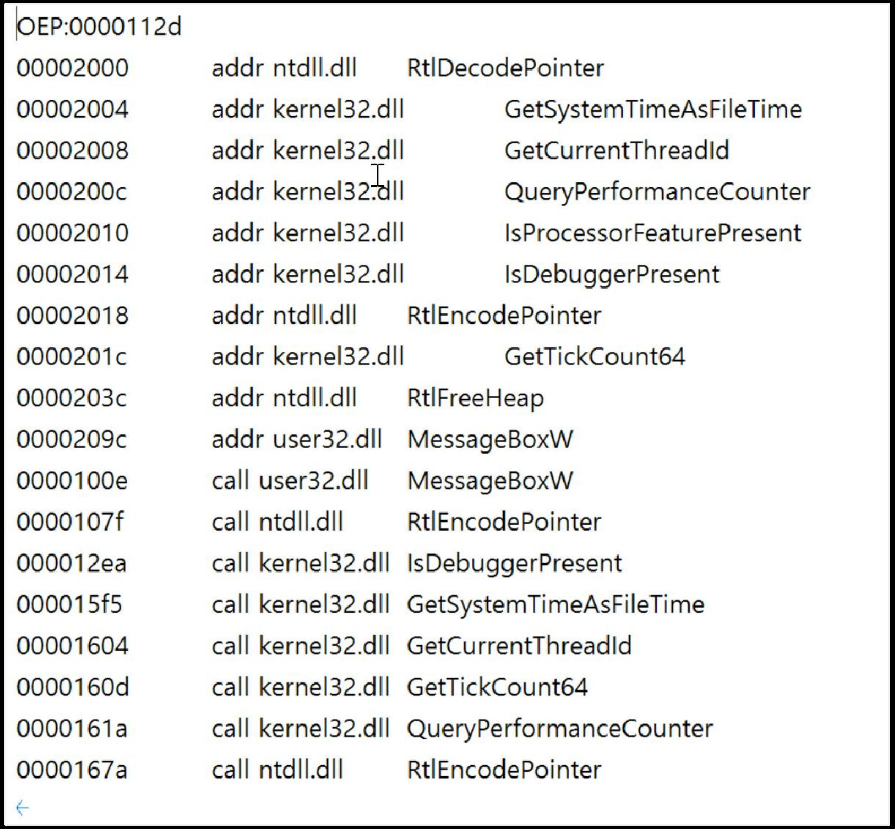
- Identifying obfuscated calls that use indirect address
  - Some call instructions use register indirect calls ex) call EDX
  - Those registers are assigned with obfuscated API address in IAT
  - But original segments (.text, .idata, …) are merged into one segment

Identifying Obfuscated API Call

- Identify a memory block that contains successive obfuscated API function addresses
- Save IAT resolution information that maps referenced addresses to original API function name

Identifying Obfuscated API Call

- Example: Generated text file



```
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

Resolving Obfuscated API Call

- How to debug obfuscated binary?
  - Use a debugger to execute a packed binary until OEP and patch obfuscated API call addresses
  - Use the pin tool to execute a packed binary until OEP and attach a debugger to the process

Resolving Obfuscated API Call

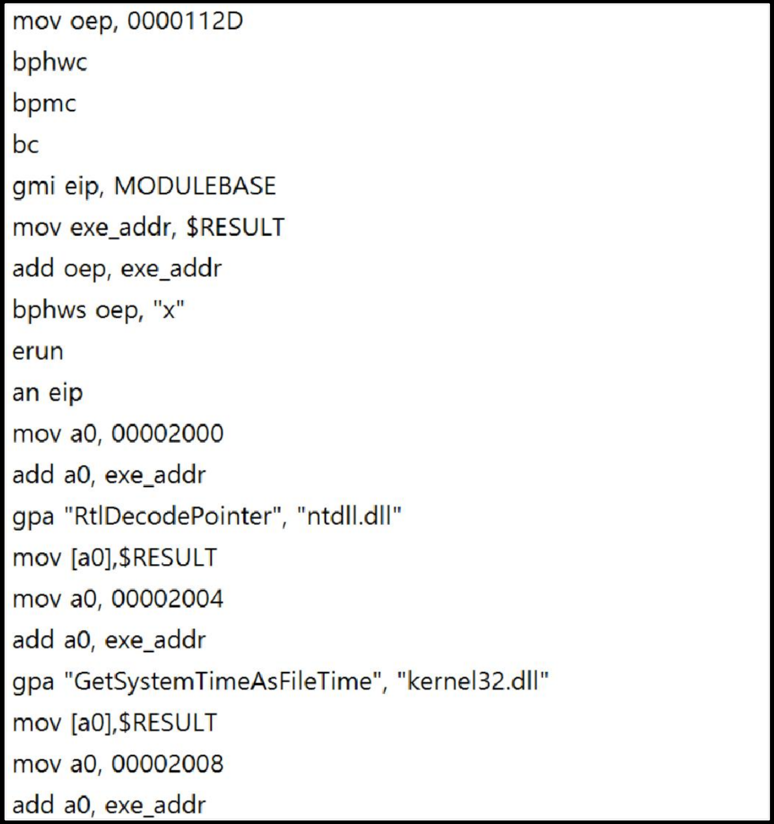
- Attaching a debugger to the obfuscated process
  - Implement anti-anti-attach techniques to the analyzer
    - Protect ntdll.DBGUiRemoteBreakin and ntdll.DBGBreakpoiont from patching
    - Prevent executing ntdll.NtSetInformationThread setting ThreadHideFromDebugger flag
  - Need to disarm monitoring threads

Resolving Obfuscated API Call

- 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 an ODBG script that execute until OEP and resolve obfuscated addresses

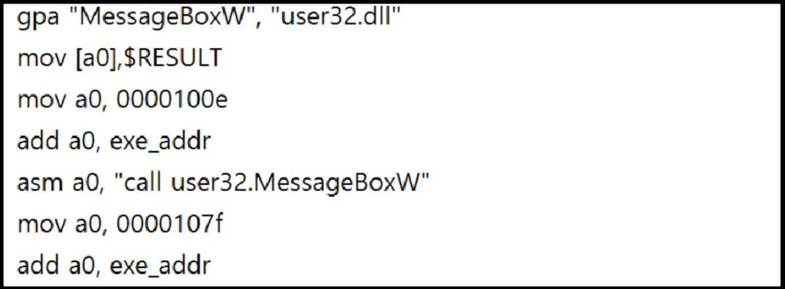
Resolving Obfuscated API Call

- ODBGScript Example



```
mov oep, 0000112D
bphwc
bpmc
bc
gmi eip, MODULEBASE
mov exe_addr, $RESULT
add oep, exe_addr
bphws oep, "x"
erun
an eip
mov a0, 00002000
add a0, exe_addr
gpa "RtlDecodePointer", "ntdll.dll"
mov [a0],$RESULT
mov a0, 00002004
add a0, exe_addr
gpa "GetSystemTimeAsFileTime", "kernel32.dll"
mov [a0],$RESULT
mov a0, 00002008
add a0, exe_addr
```

……



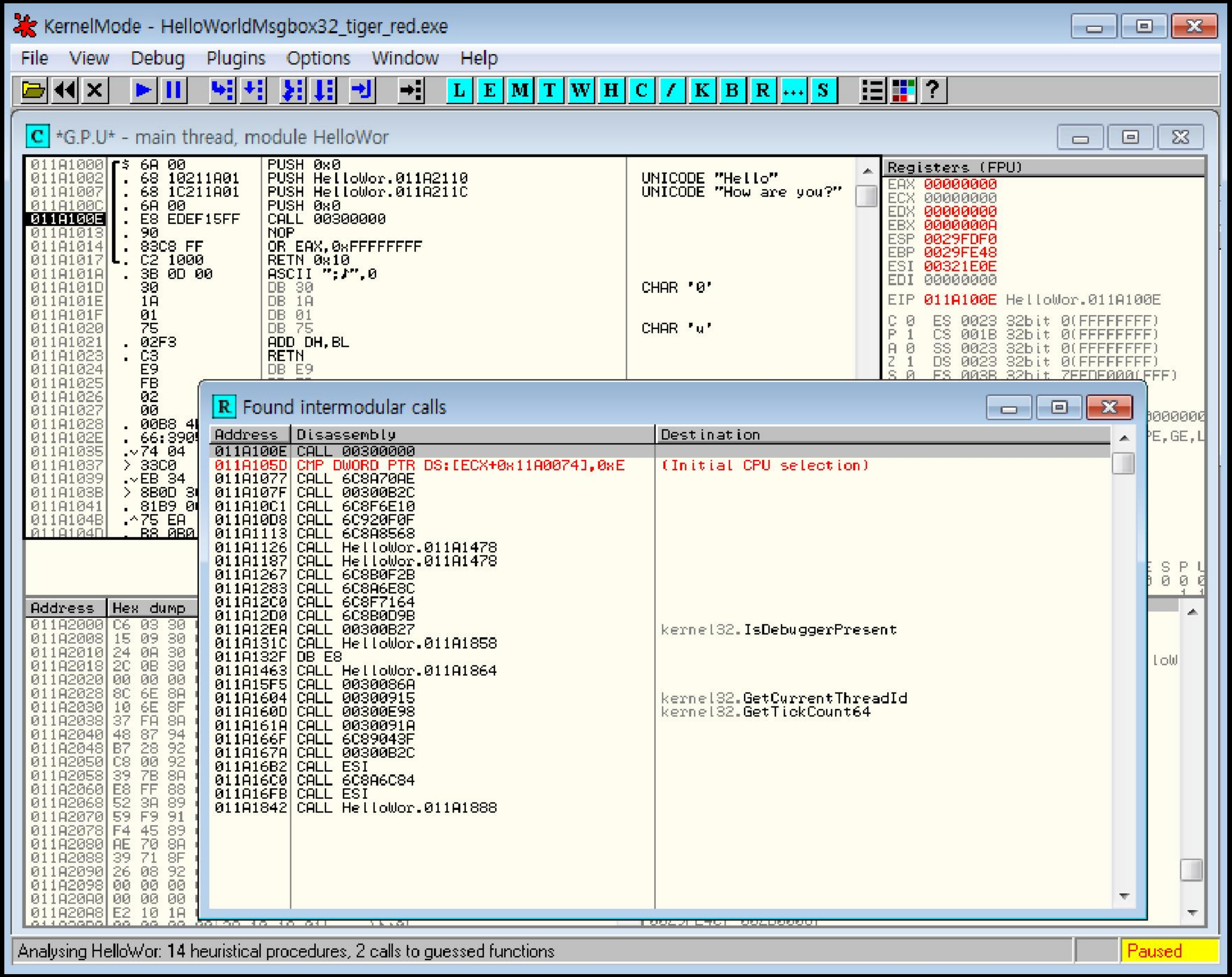
```
gpa "MessageBoxW", "user32.dll"
mov [a0],$RESULT
mov a0, 0000100e
add a0, exe_addr
asm a0, "call user32.MessageBoxW"
mov a0, 0000107f
add a0, exe_addr
```

……

Implementation

- Memory access analyzer
  - OEP Detector + API call resolver
  - Built as a pin tool (VC 2013, Intel pin 2.14)
  - Works well on Windows 7/8/8.1 x86/64
  - Anti-anti-attach capability to attach a debugger
- ODBGScript generator
  - A python script to generate ODBGScript that execute until OEP and resolve obfuscated API addresses

Debugging Obfuscated Binary


KernelMode - HelloWorldMsgbox32_tiger_red.exe
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
C *G.P.U* - main thread, module HelloWor
R Found intermodular calls
Paused

```
011A1000 ┌$ 6A 00          PUSH 0x0
011A1002 │. 68 10211A01    PUSH HelloWor.011A2110          UNICODE "Hello"
011A1007 │. 68 1C211A01    PUSH HelloWor.011A211C          UNICODE "How are you?"
011A100C │. 6A 00          PUSH 0x0
011A100E │. E8 EDEF15FF    CALL 00300000
011A1013 │. 90             NOP
011A1014 │. 83C8 FF        OR EAX,0xFFFFFFFF
011A1017 └. C2 1000        RETN 0x10
011A101A  . 3B 0D 00       ASCII ";♪",0
011A101D    30             DB 30                           CHAR '0'
011A101E    1A             DB 1A
011A101F    01             DB 01
011A1020    75             DB 75                           CHAR 'u'
011A1021  . 02F3           ADD DH,BL
011A1023  . C3             RETN
011A1024    E9             DB E9
011A1025    FB
011A1026    02
011A1027    00
011A1028  . 00B8 4
011A102E  . 66:3905
011A1035  .v74 04
011A1037  > 33C0
011A1039  .vEB 34
011A103B  > 8B0D 3
011A1041  . 81B9 0
011A104B  .^75 EA
011A104D  . B8 0B0
```

```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 00000000
EBX 0000000A
ESP 0029FDF0
EBP 0029FE48
ESI 00321E0E
EDI 00000000

EIP 011A100E HelloWor.011A100E

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)
```

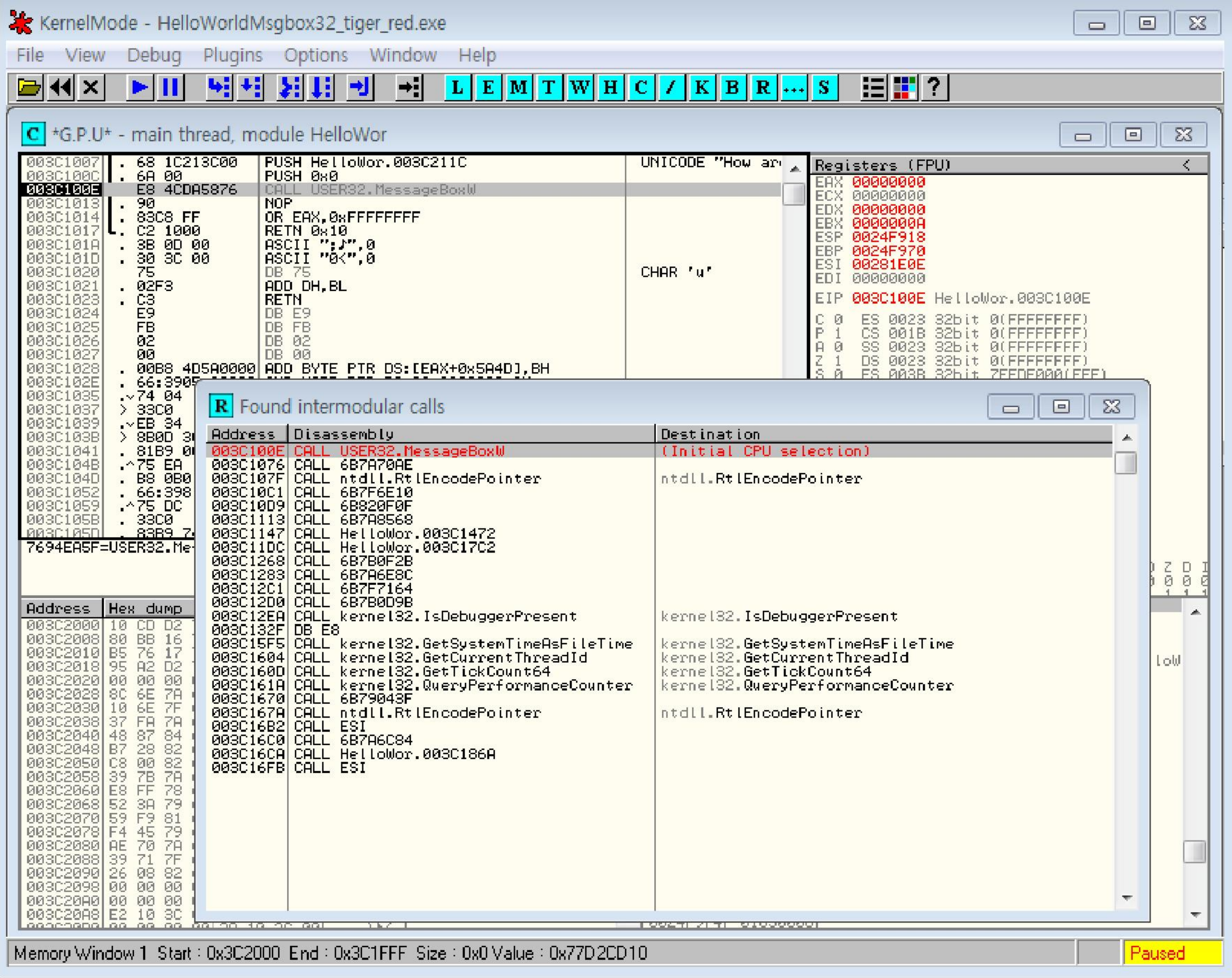
```
Address  Disassembly                               Destination
011A100E CALL 00300000
011A105D CMP DWORD PTR DS:[ECX+0x11A0074],0xE      (Initial CPU selection)
011A1077 CALL 6C8A70AE
011A107F CALL 00300B2C
011A10C1 CALL 6C8F6E10
011A10D8 CALL 6C920F0F
011A1113 CALL 6C8A8568
011A1126 CALL HelloWor.011A1478
011A1187 CALL HelloWor.011A1478
011A1267 CALL 6C8B0F2B
011A1283 CALL 6C8A6E8C
011A12C0 CALL 6C8F7164
011A12D0 CALL 6C8B0D9B
011A12EA CALL 00300B27                             kernel32.IsDebuggerPresent
011A131C CALL HelloWor.011A1858
011A132F DB E8
011A1463 CALL HelloWor.011A1864
011A15F5 CALL 0030086A
011A1604 CALL 00300915                             kernel32.GetCurrentThreadId
011A160D CALL 00300E98                             kernel32.GetTickCount64
011A161A CALL 0030091A
011A166F CALL 6C89043F
011A167A CALL 00300B2C
011A16B2 CALL ESI
011A16C0 CALL 6C8A6C84
011A16FB CALL ESI
011A1842 CALL HelloWor.011A1888
```

```
Address  Hex dump
011A2000 C6 03 30
011A2008 15 09 30
011A2010 24 0A 30
011A2018 2C 0B 30
011A2020 00 00 00
011A2028 8C 6E 8A
011A2030 10 6E 8F
011A2038 37 FA 8A
011A2040 48 87 94
011A2048 B7 28 92
011A2050 C8 00 92
011A2058 39 7B 8A
011A2060 E8 FF 88
011A2068 52 3A 89
011A2070 59 F9 91
011A2078 F4 45 89
011A2080 AE 70 8A
011A2088 39 71 8F
011A2090 26 08 92
011A2098 00 00 00
011A20A0 00 00 00
011A20A8 E2 10 1A
```

Analysing HelloWor: 14 heuristical procedures, 2 calls to guessed functions

Before deobfuscation after unpack

Debugging Obfuscated Binary


KernelMode - HelloWorldMsgbox32_tiger_red.exe
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
C *G.P.U* - main thread, module HelloWor
R Found intermodular calls

```
003C1007 | . 68 1C213C00    PUSH HelloWor.003C211C          UNICODE "How ar
003C100C | . 6A 00          PUSH 0x0
003C100E |   E8 4CDA5876    CALL USER32.MessageBoxW
003C1013 | . 90             NOP
003C1014 | . 83C8 FF        OR EAX,0xFFFFFFFF
003C1017 L. C2 1000         RETN 0x10
003C101A   . 3B 0D 00       ASCII ";♪",0
003C101D   . 30 3C 00       ASCII "0<",0
003C1020     75             DB 75                           CHAR 'u'
003C1021   . 02F3           ADD DH,BL
003C1023   . C3             RETN
003C1024     E9             DB E9
003C1025     FB             DB FB
003C1026     02             DB 02
003C1027     00             DB 00
003C1028   . 00B8 4D5A0000  ADD BYTE PTR DS:[EAX+0x5A4D],BH
003C102E   . 66:3905
003C1035   .v74 04
003C1037   > 33C0
003C1039   .vEB 34
003C103B   > 8B0D 3
003C1041   . 81B9 0
003C104B   .^75 EA
003C104D   . B8 0B0
003C1052   . 66:398
003C1059   .^75 DC
003C105B   . 33C0
003C105D   . 83B9 7
7694EA5F=USER32.Me
```

```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 00000000
EBX 0000000A
ESP 0024F918
EBP 0024F970
ESI 00281E0E
EDI 00000000

EIP 003C100E HelloWor.003C100E

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 7EFDE000(FFF)
```

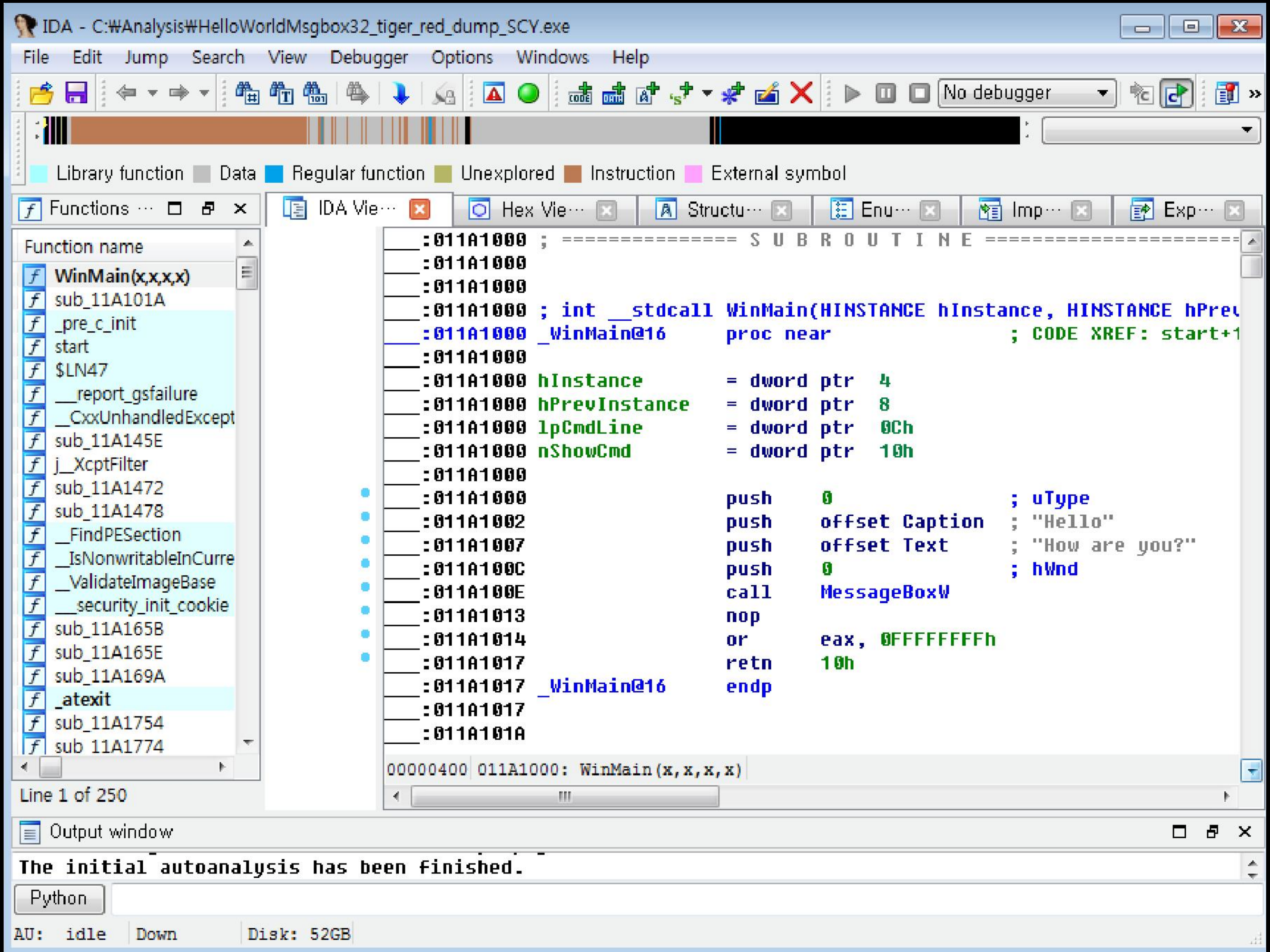
```
Address  Hex dump
003C2000 10 CD D2
003C2008 80 BB 16
003C2010 B5 76 17
003C2018 95 A2 D2
003C2020 00 00 00
003C2028 8C 6E 7A
003C2030 10 6E 7F
003C2038 37 FA 7A
003C2040 48 87 84
003C2048 B7 28 82
003C2050 C8 00 82
003C2058 39 7B 7A
003C2060 E8 FF 78
003C2068 52 3A 79
003C2070 59 F9 81
003C2078 F4 45 79
003C2080 AE 70 7A
003C2088 39 71 7F
003C2090 26 08 82
003C2098 00 00 00
003C20A0 00 00 00
003C20A8 E2 10 3C
```

Address	Disassembly	Destination
003C100E	CALL USER32.MessageBoxW	(Initial CPU selection)
003C1076	CALL 6B7A70AE	
003C107F	CALL ntdll.RtlEncodePointer	ntdll.RtlEncodePointer
003C10C1	CALL 6B7F6E10	
003C10D9	CALL 6B820F0F	
003C1113	CALL 6B7A8568	
003C1147	CALL HelloWor.003C1472	
003C11DC	CALL HelloWor.003C17C2	
003C1268	CALL 6B7B0F2B	
003C1283	CALL 6B7A6E8C	
003C12C1	CALL 6B7F7164	
003C12D0	CALL 6B7B0D9B	
003C12EA	CALL kernel32.IsDebuggerPresent	kernel32.IsDebuggerPresent
003C132F	DB E8	
003C15F5	CALL kernel32.GetSystemTimeAsFileTime	kernel32.GetSystemTimeAsFileTime
003C1604	CALL kernel32.GetCurrentThreadId	kernel32.GetCurrentThreadId
003C160D	CALL kernel32.GetTickCount64	kernel32.GetTickCount64
003C161A	CALL kernel32.QueryPerformanceCounter	kernel32.QueryPerformanceCounter
003C1670	CALL 6B79043F	
003C167A	CALL ntdll.RtlEncodePointer	ntdll.RtlEncodePointer
003C16B2	CALL ESI	
003C16C0	CALL 6B7A6C84	
003C16CA	CALL HelloWor.003C186A	
003C16FB	CALL ESI	

Memory Window 1 Start : 0x3C2000 End : 0x3C1FFF Size : 0x0 Value : 0x77D2CD10 | Paused

After resolving obfuscated addresses,
Original API call is recovered

Analyzing Deobufscated File



IDA - C:₩Analysis₩HelloWorldMsgbox32_tiger_red_dump_SCY.exe

Function name

- WinMain(x,x,x,x)
- sub_11A101A
- _pre_c_init
- start
- $LN47
- __report_gsfailure
- __CxxUnhandledExcept
- sub_11A145E
- j__XcptFilter
- sub_11A1472
- sub_11A1478
- __FindPESection
- __IsNonwritableInCurre
- __ValidateImageBase
- __security_init_cookie
- sub_11A165B
- sub_11A165E
- sub_11A169A
- _atexit
- sub_11A1754
- sub_11A1774

Line 1 of 250

```
:011A1000 ; =============== S U B R O U T I N E =======================
:011A1000
:011A1000
:011A1000 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrev
:011A1000 _WinMain@16     proc near               ; CODE XREF: start+1
:011A1000
:011A1000 hInstance       = dword ptr  4
:011A1000 hPrevInstance   = dword ptr  8
:011A1000 lpCmdLine       = dword ptr  0Ch
:011A1000 nShowCmd        = dword ptr  10h
:011A1000
:011A1000                 push    0               ; uType
:011A1002                 push    offset Caption  ; "Hello"
:011A1007                 push    offset Text     ; "How are you?"
:011A100C                 push    0               ; hWnd
:011A100E                 call    MessageBoxW
:011A1013                 nop
:011A1014                 or      eax, 0FFFFFFFFh
:011A1017                 retn    10h
:011A1017 _WinMain@16     endp
:011A1017
:011A101A
```

```
00000400 011A1000: WinMain(x,x,x,x)
```

```
The initial autoanalysis has been finished.
```

```
AU:  idle   Down      Disk: 52GB
```

Disassembled by IDA on dumped file

Related Work

- Obfuscation pattern based approach
  - Themida/Winlicense Ultra Unpacker 1.4
    - ODBGScript to unpack Themida & Winlicense file
    - Need to understand whole script to fix problems
    - Need new version when obfuscation pattern changed
- Optimization based approach
  - Possible to optimize dynamic instruction traces
  - Hard to get the whole function code because of anti-disassembly

Related Work

- Deobfuscator for virtualization-obfuscation
  - Backward slicing on API parameters - Koogan et al. (CCS ’11)
  - Taint analysis to recover CFG – B. Yadegari et al. (S&P’15)
  - Optimizing code by clustering – J. Raber (BH USA ’13)

Limitation

- DBI detection is possible
  - Execution behavior is different (BH USA'14 Defeating the transparency feature of DBI)
- Memory access pattern can be changed
  - Obfuscators can alter memory access patterns
- Unable to detect API function obfuscated by virtualization macro

Future Work

- Building deobfuscator based on emulators
  - — Avoid DBI detection
- Resolving virtualization obfuscated API calls
  - — Statically identify API calls by code emulation
  - — Utilize dynamic trace to resolve executed API calls

Demo

- Obfuscated Malware Analysis
  - Environment
    - Windows 7 x86 on VMWare
    - Pin 2.14
    - OllyDBG 1.10 with StrongOD, Phant0m
  - Debugging
  - Disassembling (decompiling)