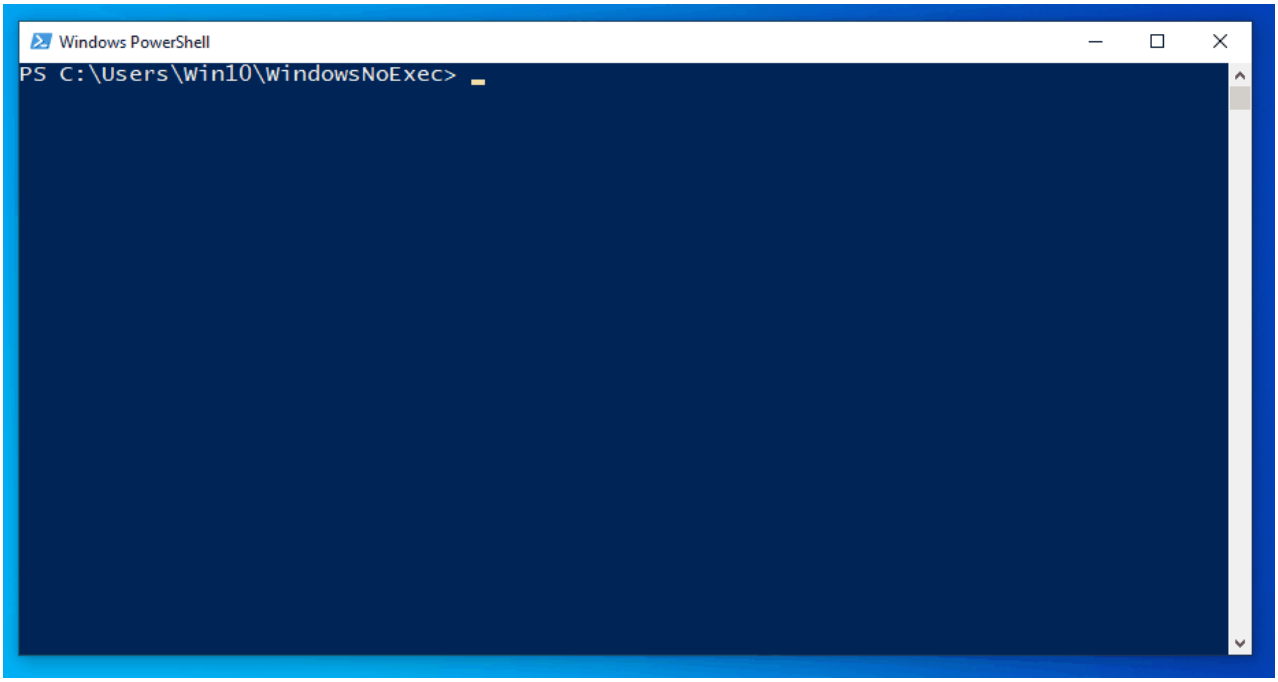
x86matthew - WindowsNoExec - Abusing existing instructions to executing arbitrary code without allocating executable memory

x86matthew.com/view_post

WindowsNoExec - Abusing existing instructions to executing arbitrary code without allocating executable memory

This proof-of-concept allows us to "re-use" existing instructions in ntdll.dll to execute our own code. The target code only exists within the data section, which means this method circumvents non-executable memory protections. The exception handler code does need to be present in the target process, but the payload can be variable. This has limited use in the real world as custom exception handlers are very easy to detect, but it might make an interesting CTF challenge for learning purposes.

Interestingly, this technique only requires one API - RtlAddVectoredExceptionHandler. My code also uses GetModuleHandle to retrieve the base address of ntdll.dll, but this is an easy function to re-create if necessary.



This code works as follows:
1. Create a data structure containing all of the assembly instructions that we want to execute.
2. Search the code section of ntdll.dll for each of the instructions above and store the addresses.
3. Add a custom exception handler in our program using RtlAddVectoredExceptionHandler.
4. Trigger a breakpoint using int 3.
5. The program has now entered our custom exception handler. Store the original thread

context for later.
6. Set the EIP register to the first target instruction (within ntdll.dll) in our list.
7. If the current instruction is a 'call', set a hardware-breakpoint using the Dr0 debug register on the instruction directly after the call - we want to "step over" the call. Otherwise, set the single-step flag using EFlags |= 0x100 to break on the next instruction.
8. Update the values of any other registers that are necessary for the current instruction.
9. Continue execution using EXCEPTION_CONTINUE_EXECUTION. The next instruction will raise another exception, and we will continue from step #6 until all of the instructions have been run in sequence.
10. After all of the target instructions have been executed, restore the original thread context from step #5 to continue the original flow of the program.

The following data structure will call MessageBoxA:

```
InstructionEntryStruct Global_InstructionList[] =
{
// allocate 1kb buffer for messagebox title using GlobalAlloc
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 1024, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, GMEM_FIXED, 0, 0, 0, FLAG_ECX },
{ "call eax ; (GlobalAlloc)", { 0xFF, 0xD0 }, 2, 0, (DWORD)GlobalAlloc, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },

// set messagebox title to "www.x86matthew.com"
{ "mov ebx, eax", { 0x8B, 0xD8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '.' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '.', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'x' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'x', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '8' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '8', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '6' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '6', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
```

```
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'a' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'a', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'h' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'h', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '.' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '.', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'c' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'c', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'o' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'o', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; (null) ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '\0', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

// store messagebox title ptr in edi register
{ "mov edi, eax", { 0x8B, 0xF8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },

// allocate 1kb buffer for messagebox text using GlobalAlloc
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 1024, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, GMEM_FIXED, 0, 0, 0, FLAG_ECX },
{ "call eax ; (GlobalAlloc)", { 0xFF, 0xD0 }, 2, 0, (DWORD)GlobalAlloc, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },

// set messagebox text to "A message box from ntdll.dll"
{ "mov ebx, eax", { 0x8B, 0xD8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'A' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'A', 0, 0, FLAG_EDX
```

```
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 's' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 's', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 's' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 's', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'a' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'a', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'g' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'g', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'b' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'b', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'o' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'o', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'x' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'x', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'f' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'f', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'r' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'r', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'o' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'o', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
```

```
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'n' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'n', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'd' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'd', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '.' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '.', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'd' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'd', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; (null) ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '\0', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

// call MessageBoxA
{ "push ecx", { 0x51 }, 1, 0, 0, 0, MB_OK, 0, 0, 0, FLAG_ECX },
{ "push edi", { 0x57 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "push eax", { 0x50 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 0, 0, 0, 0, FLAG_ECX },
{ "call eax ; (MessageBoxA)", { 0xFF, 0xD0 }, 2, 0, (DWORD)MessageBoxA, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },
};
```

The structure header contains the following fields:

```
struct InstructionEntryStruct
{
char *pLabel;
```

```
BYTE bInstruction[16];
DWORD dwInstructionLength;

DWORD dwInstructionAddr;

DWORD dwEax;
DWORD dwEbx;
DWORD dwEcx;
DWORD dwEdx;
DWORD dwEdi;
DWORD dwEsi;
DWORD dwInstructionFlags;
};
```

pLabel
This field is for logging/debugging purposes only.

bInstruction
This field contains the opcode of the target instruction - eg 0x50 for push eax.

dwInstructionLength
The length of the bInstruction field.

dwInstructionAddr
This field is populated by the program - ntdll.dll is scanned to find the address of a matching instruction.

dwEax / dwEbx / dwEcx / dwEdx / dwEdi / dwEsi
These fields set the specified register values before the current instruction is executed.

dwInstructionFlags
This field specifies which register values should be updated (see above). It is also used to specify whether or not the current instruction is a 'call'.

It is important to be careful with the instruction opcodes that we pick. For example, if we wanted to include a push 0x12345678 instruction, we could do the following:

```
{ "push 0x12345678", { 0x68, 0x44, 0x33, 0x22, 0x11 }, 5, 0, 0, 0, 0, 0, 0, 0, 0 }
```

... but this would not work. This is because ntdll.dll is very unlikely to contain a sequence containing [0x68, 0x44, 0x33, 0x22, 0x11] in the code section. This code takes advantage of the fact that we can manipulate the registers within the exception handler before the instruction is executed, which means that we can do the following instead:

```
{ "push eax", { 0x50 }, 1, 0, 0x11223344, 0, 0, 0, 0, 0, FLAG_EAX }
```

The entry above only relies on a single 0x50 (push eax) byte being found in the ntdll.dll code section. The eax register value will be set to 0x11223344 by the exception handler prior to the instruction being executed.

The following data structure demonstrates a slightly more complicated example - this creates a file (x86matthew.txt), and writes "Text file created by ntdll" to the returned handle:

```
InstructionEntryStruct Global_InstructionList[] =
{
// allocate 1kb buffer for filename using GlobalAlloc
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 1024, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, GMEM_FIXED, 0, 0, 0, FLAG_ECX },
{ "call eax ; (GlobalAlloc)", { 0xFF, 0xD0 }, 2, 0, (DWORD)GlobalAlloc, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },

// set filename to "x86matthew.txt"
{ "mov ebx, eax", { 0x8B, 0xD8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'x' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'x', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '8' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '8', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '6' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '6', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'a' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'a', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'h' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'h', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
```

```
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '.' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '.', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'x' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'x', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; (null) ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '\0', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

// call CreateFileA
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 0, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, FILE_ATTRIBUTE_NORMAL, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, CREATE_ALWAYS, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 0, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 0, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, GENERIC_WRITE, 0, 0, 0, FLAG_ECX },
{ "push eax", { 0x50 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "call eax ; (CreateFileA)", { 0xFF, 0xD0 }, 2, 0, (DWORD)CreateFileA, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },

// store file handle in esi register
{ "mov esi, eax", { 0x8B, 0xF0 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 }, // mov esi, eax (esi=hFile)

// allocate 1kb buffer for file content using GlobalAlloc
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 1024, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, GMEM_FIXED, 0, 0, 0, FLAG_ECX },
{ "call eax ; (GlobalAlloc)", { 0xFF, 0xD0 }, 2, 0, (DWORD)GlobalAlloc, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },

// set file content buffer to "Text file created by ntdll"
{ "mov ebx, eax", { 0x8B, 0xD8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'T' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'T', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
```

```
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'x' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'x', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'f' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'f', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'i' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'i', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'c' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'c', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'r' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'r', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'a' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'a', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'd' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'd', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'b' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'b', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'y' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'y', 0, 0, FLAG_EDX
},
```

```
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'n' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'n', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'd' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'd', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; (null) ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '\0', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

// call WriteFile (and allocate a local variable on the stack for the lpNumberOfBytesWritten
value)
{ "push ebp", { 0x55 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "push 0", { 0x6A, 0x00 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov ebp, esp", { 0x8B, 0xEC }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 0, 0, 0, 0, FLAG_ECX },
{ "push ebp", { 0x55 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 26, 0, 0, 0, FLAG_ECX },
{ "push eax", { 0x50 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "push esi", { 0x56 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "call eax ; (WriteFile)", { 0xFF, 0xD0 }, 2, 0, (DWORD)WriteFile, 0, 0, 0, 0, 0, FLAG_EAX
| FLAG_CALL },
{ "pop ecx", { 0x59 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "pop ebp", { 0x5D }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

// call CloseHandle
{ "push esi", { 0x56 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "call eax ; (CloseHandle)", { 0xFF, 0xD0 }, 2, 0, (DWORD)CloseHandle, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },
};
```

Full code below:

```
#include <stdio.h>
#include <windows.h>
```

```
#define FLAG_EAX 0x00000001
#define FLAG_EBX 0x00000002
#define FLAG_ECX 0x00000004
#define FLAG_EDX 0x00000008
#define FLAG_EDI 0x00000010
#define FLAG_ESI 0x00000020
#define FLAG_CALL 0x00000040

struct InstructionEntryStruct
{
char *pLabel;

BYTE bInstruction[16];
DWORD dwInstructionLength;

DWORD dwInstructionAddr;

DWORD dwEax;
DWORD dwEbx;
DWORD dwEcx;
DWORD dwEdx;
DWORD dwEdi;
DWORD dwEsi;
DWORD dwInstructionFlags;
};

DWORD dwGlobal_CurrInstruction = 0;
CONTEXT Global_OrigContext;

InstructionEntryStruct Global_InstructionList[] =
{
// allocate 1kb buffer for messagebox title using GlobalAlloc
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 1024, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, GMEM_FIXED, 0, 0, 0, FLAG_ECX },
{ "call eax ; (GlobalAlloc)", { 0xFF, 0xD0 }, 2, 0, (DWORD)GlobalAlloc, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },

// set messagebox title to "www.x86matthew.com"
{ "mov ebx, eax", { 0x8B, 0xD8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
```

```
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '.' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '.', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'x' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'x', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '8' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '8', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '6' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '6', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'a' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'a', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'h' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'h', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'w' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'w', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '.' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '.', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'c' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'c', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'o' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'o', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; (null) ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '\0', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
```

```
// store messagebox title ptr in edi register
{ "mov edi, eax", { 0x8B, 0xF8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },

// allocate 1kb buffer for messagebox text using GlobalAlloc
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 1024, 0, 0, 0, FLAG_ECX },
{ "push ecx", { 0x51 }, 1, 0, 0, 0, GMEM_FIXED, 0, 0, 0, FLAG_ECX },
{ "call eax ; (GlobalAlloc)", { 0xFF, 0xD0 }, 2, 0, (DWORD)GlobalAlloc, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },

// set messagebox text to "A message box from ntdll.dll"
{ "mov ebx, eax", { 0x8B, 0xD8 }, 2, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'A' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'A', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 's' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 's', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 's' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 's', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'a' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'a', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'g' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'g', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'e' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'e', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'b' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'b', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'o' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'o', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
```

```
{ "mov byte ptr [ebx], dl ; character: 'x' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'x', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'f' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'f', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'r' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'r', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'o' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'o', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'm' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'm', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: ' ' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, ' ', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'n' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'n', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 't' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 't', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'd' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'd', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: '.' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '.', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'd' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'd', 0, 0, FLAG_EDX
},
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; character: 'l' ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, 'l', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "mov byte ptr [ebx], dl ; (null) ", { 0x88, 0x13 }, 2, 0, 0, 0, 0, '\0', 0, 0, FLAG_EDX },
{ "inc ebx", { 0x43 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

// call MessageBoxA
{ "push ecx", { 0x51 }, 1, 0, 0, 0, MB_OK, 0, 0, 0, FLAG_ECX },
{ "push edi", { 0x57 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
{ "push eax", { 0x50 }, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
```

```
{ "push ecx", { 0x51 }, 1, 0, 0, 0, 0, 0, 0, 0, FLAG_ECX },
{ "call eax ; (MessageBoxA)", { 0xFF, 0xD0 }, 2, 0, (DWORD)MessageBoxA, 0, 0, 0, 0, 0,
FLAG_EAX | FLAG_CALL },
};

LONG WINAPI ExceptionHandler(EXCEPTION_POINTERS *pExceptionInfo)
{
InstructionEntryStruct *pCurrInstruction = NULL;

// ensure this is a breakpoint / single step exception
if(pExceptionInfo->ExceptionRecord->ExceptionCode != STATUS_BREAKPOINT &&
pExceptionInfo->ExceptionRecord->ExceptionCode != STATUS_SINGLE_STEP)
{
// this is not the exception that we expected - pass this exception to the next handler
return EXCEPTION_CONTINUE_SEARCH;
}

// reset hardware breakpoints
pExceptionInfo->ContextRecord->Dr0 = 0;
pExceptionInfo->ContextRecord->Dr7 = 0;

if(dwGlobal_CurrInstruction == 0)
{
// store original context
memcpy((void*)&Global_OrigContext, (void*)pExceptionInfo->ContextRecord,
sizeof(CONTEXT));
}
else if(dwGlobal_CurrInstruction >= (sizeof(Global_InstructionList) /
sizeof(Global_InstructionList[0])))
{
// finished executing all instructions - restore original context
memcpy((void*)pExceptionInfo->ContextRecord, (void*)&Global_OrigContext,
sizeof(CONTEXT));

// move to the next instruction (after int3)
pExceptionInfo->ContextRecord->Eip++;

// continue execution
return EXCEPTION_CONTINUE_EXECUTION;
}

// get current instruction entry
pCurrInstruction = &Global_InstructionList[dwGlobal_CurrInstruction];

// set instruction ptr to next instruction
pExceptionInfo->ContextRecord->Eip = pCurrInstruction->dwInstructionAddr;
```

```
// check register flags
if(pCurrInstruction->dwInstructionFlags & FLAG_EAX)
{
// set eax
printf("<InternalExHandler> mov eax, 0x%x\n", pCurrInstruction->dwEax);
pExceptionInfo->ContextRecord->Eax = pCurrInstruction->dwEax;
}
else if(pCurrInstruction->dwInstructionFlags & FLAG_EBX)
{
// set ebx
printf("<InternalExHandler> mov ebx, 0x%x\n", pCurrInstruction->dwEbx);
pExceptionInfo->ContextRecord->Ebx = pCurrInstruction->dwEbx;
}
else if(pCurrInstruction->dwInstructionFlags & FLAG_ECX)
{
// set ecx
printf("<InternalExHandler> mov ecx, 0x%x\n", pCurrInstruction->dwEcx);
pExceptionInfo->ContextRecord->Ecx = pCurrInstruction->dwEcx;
}
else if(pCurrInstruction->dwInstructionFlags & FLAG_EDX)
{
// set edx
printf("<InternalExHandler> mov edx, 0x%x\n", pCurrInstruction->dwEdx);
pExceptionInfo->ContextRecord->Edx = pCurrInstruction->dwEdx;
}
else if(pCurrInstruction->dwInstructionFlags & FLAG_EDI)
{
// set edi
printf("<InternalExHandler> mov edi, 0x%x\n", pCurrInstruction->dwEdi);
pExceptionInfo->ContextRecord->Edi = pCurrInstruction->dwEdi;
}
else if(pCurrInstruction->dwInstructionFlags & FLAG_ESI)
{
// set esi
printf("<InternalExHandler> mov esi, 0x%x\n", pCurrInstruction->dwEsi);
pExceptionInfo->ContextRecord->Esi = pCurrInstruction->dwEsi;
}

// print current instruction label
printf("<ntdll: 0x%08X> %s\n", pCurrInstruction->dwInstructionAddr, pCurrInstruction-
>pLabel);

// check if this is a 'call' instruction
if(pCurrInstruction->dwInstructionFlags & FLAG_CALL)
{
```

```
// set a hardware breakpoint on the first instruction after the 'call'
pExceptionInfo->ContextRecord->Dr0 = pCurrInstruction->dwInstructionAddr +
pCurrInstruction->dwInstructionLength;
pExceptionInfo->ContextRecord->Dr7 = 1;
}
else
{
// single step
pExceptionInfo->ContextRecord->EFlags |= 0x100;
}

// move to the next instruction
dwGlobal_CurrInstruction++;

// continue execution
return EXCEPTION_CONTINUE_EXECUTION;
}

DWORD GetModuleCodeSection(DWORD dwModuleBase, DWORD
*pdwCodeSectionStart, DWORD *pdwCodeSectionLength)
{
IMAGE_DOS_HEADER *pDosHeader = NULL;
IMAGE_NT_HEADERS *pNtHeader = NULL;
IMAGE_SECTION_HEADER *pCurrSectionHeader = NULL;
char szCurrSectionName[16];
DWORD dwFound = 0;
DWORD dwCodeSectionStart = 0;
DWORD dwCodeSectionLength = 0;

// get dos header ptr (start of module)
pDosHeader = (IMAGE_DOS_HEADER*)dwModuleBase;
if(pDosHeader->e_magic != IMAGE_DOS_SIGNATURE)
{
return 1;
}

// get nt header ptr
pNtHeader = (IMAGE_NT_HEADERS*)((BYTE*)pDosHeader + pDosHeader->e_lfanew);
if(pNtHeader->Signature != IMAGE_NT_SIGNATURE)
{
return 1;
}

// loop through all sections
for(DWORD i = 0; i < pNtHeader->FileHeader.NumberOfSections; i++)
{
```

```
// get current section header
pCurrSectionHeader = (IMAGE_SECTION_HEADER*)((BYTE*)pNtHeader +
sizeof(IMAGE_NT_HEADERS) + (i * sizeof(IMAGE_SECTION_HEADER)));

// pCurrSectionHeader->Name is not null terminated if all 8 characters are used - copy it
to a larger local buffer
memset(szCurrSectionName, 0, sizeof(szCurrSectionName));
memcpy(szCurrSectionName, pCurrSectionHeader->Name, sizeof(pCurrSectionHeader-
>Name));

// check if this is the main code section
if(strcmp(szCurrSectionName, ".text") == 0)
{
// found code section
dwFound = 1;
dwCodeSectionStart = dwModuleBase + pCurrSectionHeader->VirtualAddress;
dwCodeSectionLength = pCurrSectionHeader->SizeOfRawData;

break;
}
}

// ensure the code section was found
if(dwFound == 0)
{
return 1;
}

// store values
*pdwCodeSectionStart = dwCodeSectionStart;
*pdwCodeSectionLength = dwCodeSectionLength;

return 0;
}

DWORD ScanForInstructions()
{
DWORD dwInstructionCount = 0;
DWORD dwCurrSearchPos = 0;
DWORD dwBytesRemaining = 0;
DWORD dwFoundAddr = 0;
DWORD dwCodeSectionStart = 0;
DWORD dwCodeSectionLength = 0;

// calculate instruction count
dwInstructionCount = sizeof(Global_InstructionList) / sizeof(Global_InstructionList[0]);
```

```
// find ntdll code section range
if(GetModuleCodeSection((DWORD)GetModuleHandle("ntdll.dll"), &dwCodeSectionStart,
&dwCodeSectionLength) != 0)
{
return 1;
}

// scan for instructions
for(DWORD i = 0; i < dwInstructionCount; i++)
{
// check if an address has already been found for this instruction
if(Global_InstructionList[i].dwInstructionAddr != 0)
{
continue;
}

// find this instruction in the ntdll code section
dwCurrSearchPos = dwCodeSectionStart;
dwBytesRemaining = dwCodeSectionLength;
dwFoundAddr = 0;
for(;;)
{
// check if the end of the code section has been reached
if(Global_InstructionList[i].dwInstructionLength > dwBytesRemaining)
{
break;
}

// check if the instruction exists here
if(memcmp((void*)dwCurrSearchPos, (void*)Global_InstructionList[i].bInstruction,
Global_InstructionList[i].dwInstructionLength) == 0)
{
dwFoundAddr = dwCurrSearchPos;
break;
}

// update search indexes
dwCurrSearchPos++;
dwBytesRemaining--;
}

// ensure the opcode was found
if(dwFoundAddr == 0)
{
printf("Error: Instruction not found in ntdll: '%s'\n", Global_InstructionList[i].pLabel);
```

```
return 1;
}

// store address
Global_InstructionList[i].dwInstructionAddr = dwFoundAddr;

// copy this instruction address to any other matching instructions in the list
for(DWORD ii = 0; ii < dwInstructionCount; ii++)
{
// check if the instruction lengths match
if(Global_InstructionList[ii].dwInstructionLength ==
Global_InstructionList[i].dwInstructionLength)
{
// check if the instruction opcodes match
if(memcmp(Global_InstructionList[ii].bInstruction, Global_InstructionList[i].bInstruction,
Global_InstructionList[i].dwInstructionLength) == 0)
{
// copy instruction address
Global_InstructionList[ii].dwInstructionAddr = Global_InstructionList[i].dwInstructionAddr;
}
}
}
}

return 0;
}

int main()
{
PVOID (WINAPI *RtlAddVectoredExceptionHandler)(DWORD dwFirstHandler, void
*pExceptionHandler);
DWORD dwThreadID = 0;
HANDLE hThread = NULL;

printf("WindowsNoExec - www.x86matthew.com\n\n");

// get RtlAddVectoredExceptionHandler function ptr
RtlAddVectoredExceptionHandler = (void *(__stdcall *)(unsigned long,void
*))GetProcAddress(GetModuleHandle("ntdll.dll"), "RtlAddVectoredExceptionHandler");
if(RtlAddVectoredExceptionHandler == NULL)
{
return 1;
}

printf("Adding exception handler...\n");
```

```
// add exception handler
if(RtlAddVectoredExceptionHandler(1, (void*)ExceptionHandler) == NULL)
{
return 1;
}

printf("Scanning ntdll to populate instruction list...\n");

// scan for instructions
if(ScanForInstructions() != 0)
{
return 1;
}

printf("Starting...\n\n");

// breakpoint to trigger exception handler
_asm int 3

printf("\nFinished\n");

return 0;
}
```