Title:

Applied
Binary Code Obfuscation

Date:

January 21st 2009

Website:

http://www.astalavista.com

Author:

Nicolaou George

Mail:

ishtus<\at>astalavista<\d0t>com

Author:

Glafkos Charalambous

Mail:

glafkos<\at>astalavista<\d0t>com

Table of Contents

Introduction

An obfuscated code is the one that is hard (but not impossible) to read and understand. Sometimes corporate developers, programmers and **malware coders** for security reasons, intentionally obfuscate their software in an attempt to delay reverse engineering or confuse antivirus engines from identifying malicious behaviors. Nowadays, obfuscation is often applied to object oriented cross-platform programming languages like Java, .NET (C#, VB), Perl, Ruby, Python and PHP. That is because their code can be easily decompiled and examined making them vulnerable to reverse engineering. On the other hand, obfuscating binary code is not as easy as encrypting object or function names as it is done in programming languages mentioned above. In this case, the code is altered by using a variety of transformations, for instance self modifying code, stack operations or even splitting the factors of simple mathematical functions. Moreover, binary obfuscation is also used to defeat automated network traffic analyzers such like Intrusion Detection and Prevention Systems. In other words, binary code obfuscation is the technique of altering the original code structure and maintaining its original functionality. In the next pages of this paper we will explore the theory and practice of binary code obfuscation as well as a number of various techniques that can be used.

Tools

The tools used in this paper are the following:

- OllyDBG [http://www.ollydbg.de/]
- WinAsm Studio [http://www.winasm.net/]

Instruction Obfuscations

Obfuscation techniques aim to replace the instructions of a binary file or any other executable code with ones that have equivalent functionalities. Sometimes the size of those instructions is of no concern, since the main goal is to render the file or network traffic unreadable and undetectable. Therefore, obfuscated programs are most likely to be bigger than the original ones.

We can distinguish the methods used in two main categories for the purposes of this paper:

- The ones that use stack operations as a mean of obfuscation (*Stack Based Obfuscation*)
- The ones that use logical and arithmetical and logical instructions as a mean of obfuscation (*Arithmetic and Logical Based Obfuscation*)

EIP (the Instruction Pointer)

For the purposes of some obfuscation techniques we will need to retrieve the value of EIP. However we cannot directly reference it (wouldn't life be easier if we could do so? I think not) thus we will have to find a way around. There are several ways of doing so but I will explain the most commonly used. Note that, the value of EIP is pushed into the stack each time you use the *CALL* instruction and we can use that to our advantage.

return EIP;

The simplest way to retrieve the value of EIP is to create a function that reads EIP and returns it.

OFFSET+00	CALL <FUNCTION>
OFFSET+05	EAX ←
FUNCTION	POP EAX
	PUSH EAX
	RETN

At this point, EAX is equal to the value of EIP (OFFSET+05)

MASM (WinAsm) Code Sample:

```
;Code Omitted

getIP PROC

        pop EAX         ;Place the value of EIP from the stack inside EAX
        push EAX        ;Push the value back to the stack so we won't affect the program flow
        retn
getIP EndP

;Code Omitted

invoke getIP ;After executing this instruction, EAX will contain the value of EIP at this point
```

CALL myself

Another way to get the value of EIP is to use a CALL function that calls itself

Offset	Bytes	Instruction
OFFSET+00	E8 00 00 00 00	CALL <MYSELF>
OFFSET+05	58	POP EAX ←

MASM (WinAsm) Code:

```
;Code Omitted

db 0e8h ;We hard-code the actual bytes inside our code
dd 00000000h
pop eax ;After executing this instruction, EAX will contain the value of EIP at this point
```

Assembled code:

```
0040102B   .   817D 0C 1101  CMP DWORD PTR SS:[EBP+C],111
00401032   .v  75 54         JNZ SHORT geteip.00401088
00401034   .   817D 10 E903  CMP DWORD PTR SS:[EBP+10],3E9
0040103B   .v  75 36         JNZ SHORT geteip.00401073
0040103D   .   E8 00000000   CALL geteip.00401042                EAX = EIP
00401042   $   58            POP EAX
00401043   .   68 04314000   PUSH geteip.00403104
00401048   .   50            PUSH EAX
00401049   .   E8 72000000   CALL geteip.004010C0
0040104E   .   68 04314000   PUSH geteip.00403104               Text = ""
```

"MOV EAX,EIP"

MASM (WinAsm) Code Sample 2:

```
;Code Omitted
mov eax,$ ; After executing this instruction, EAX will contain the value of EIP at this point, the offset of the next
            instruction therefore will be EAX+5
```

Sample 2 assembled code:

```
00401032  .v 75 53        JNZ SHORT geteip.00401087
00401034  .  817D 10 E903 CMP DWORD PTR SS:[EBP+10],3E9
0040103B  .v 75 35        JNZ SHORT geteip.00401072
0040103D  .  B8 3D104000  MOV EAX,geteip.0040103D          EAX = EIP
00401042  .  68 04314000  PUSH geteip.00403104
00401047  .  50           PUSH EAX
00401048  .  E8 73000000  CALL geteip.004010C0
0040104D  .  68 04314000  PUSH geteip.00403104             Text = ""
00401052  .  68 EB030000  PUSH 3EB                         ControlID = 3EB (1003.)
00401057  .  FF75 08      PUSH DWORD PTR SS:[EBP+8]        hWnd
```

Stack Based Obfuscation

In this category, we will explore most techniques that use the stack as a mean of obfuscation. The stack is an abstract data structure based on the principle of FIFO (First In First Out). In our case the stack is similar to the following figure:

The ESP (Stack Pointer) points to the first object inside the stack. As you can see the second item is located at ESP+4, the third at ESP+8 and so on. Once a POP instruction is executed, the first item is taken out the stack, and then the stack pointer (ESP) points at the second item. Once a PUSH instruction is executed then the pushed value is added into the stack and the stack pointer (ESP) points to it.

Stack based obfuscation manipulates the stack or references to it in order to succeed certain transformations of instructions.

1. PUSH <value/register>

We have explained above how the PUSH instruction works. What we need to do is find a way to replace this instruction with a different one that has the same functionality.

We need an instruction or a set of instructions that:

- Stores the value or the value of a register in the stack
- Makes ESP point to that value

Since we can directly affect the ESP register and reference to the stack meaning that we are able to write data to it we could easily replace the PUSH instruction with:

```
MOV DWORD PTR SS:[ESP-4],<value/register>
SUB ESP,4

or

SUB ESP,4
MOV DWORD PTR SS:[ESP],<value/register>
```

Still we can obfuscate the PUSH <value/register> instruction by storing the value to be pushed in another register or memory address

```
MOV <register/[memory]>,<value/register>
PUSH <register/[memory]>
```

2. POP <register/[memory]>

The same goes with the POP instruction. We have the stack register (ESP) currently pointing at the value we would like to retrieve from the stack and therefore we can replace POP with the following set of instructions:

We store the value currently pointed by ESP into any register then balance the Stack

```
MOV <register>,DWORD[ESP]
ADD ESP,4
```

Additionally, instead of directly POPing the value to the desired register (or memory address) we first POP it in a predefined memory address and copy it over to our register

```
POP <[memory]/>
MOV <register>,<[memory]>
```

3. CALL <address>

The CALL instruction is used for jumping into functions or system calls. In the processors level, the address of the next instruction is pushed into the stack for retrieval. After the function is finished executing and the RETN instruction is called, the value of the instruction pointer (EIP) changes and points to the address at the top of the stack.

Code:

Offset	Instruction
OFFSET+00	CALL <PROCEDURE>
OFFSET+05	<INSTRUCTION>

← OFFSET+05 is pushed into the stack and EIP changes to FUNCTION+XX

Function x:

Offset	Instruction
FUNCTION+00	<INSTRUCTION>
FUNCTION+XX	<INSTRUCTION>
FUNCTION+XX	RETN

← The first value is POPed from the stack and placed into EIP

We can replace this instruction with:

```
PUSH EIP+X      ; where X is the number of bytes until the next instruction (in the case above that is EIP+05)
JMP <address>   ; where address is the offset of the function (in the case above that is FUNCTION+00)
```

(Note that: the example above can be combined with the PUSH obfuscations explained in section 1)

Another set of instructions we can use to replace the CALL instruction is:

```
MOV <[memory]/register>,<address>
CALL <[memory]/register>
```

4. RETN

The RETN instruction is used to return from a function or a system call as explained above in section 3.

We can easily replace this instruction with:

```
POP <register/[memory]>
JMP <register/[memory]>
```

(Note that: the example above can be combined with the PUSH and POP obfuscations explained in section 1 and 2)

5. MOV <register/[memory]>,<register/[memory]/value>*

*Note that MOV <[memory]>,<[memory]> is not a valid instruction but you can still use the obfuscation set of instructions provided below to achieve it.

The MOV instruction moves the value of the right side operand to the left side operand
eg: LeftOPER = RightOPER

This can be replaced with:

```
PUSH <register/[memory]/value>    ; This is the right side operand
POP <register/[memory]>           ; This is the left side operand
```

(Note that: the example above can be combined with the PUSH and POP obfuscations explained in section 1 and 2. Also in order to POP to a memory address you are required to have the appropriate permissions)

OR in some cases:

```
LEA <register a>,DWORD[<register b>]
```

6. JMP <address>

One of the most common opcodes that are obfuscated is the JMP instruction. JMP (Jump) or any other conditional jumps (JE, JNZ, JO, JA, etc) are used for controlling program flow under certain conditions or unconditionally. Ways of obfuscating such instructions are:

```
PUSH <address>
RETN
```

(Note that: the example above can be combined with the PUSH and RETN obfuscations explained in section 1 and 4)

```
CALL <address>

(Optionally at <address>, in order to balance the stack and get rid of the return address from the stack)
POP <register>
(Or)
ADD ESP,4
```

(Note that: the example above can be combined with the CALL and POP obfuscations explained in section 3 and 2)

Conditional Jumps can be obfuscated by using the SAHF and LAHF instructions (See page 13)

Arithmetic and Logical Binary Obfuscation

The *Logical and Arithmetical* obfuscation can be considered as less technical than the *Stack Based Obfuscation.* It involves the use of simple mathematical and logical functions like ADD, SUB, AND, NOT, NEG, OR, XOR, TEST, SHL, LEA, etc.

Every mathematical function f() can be written in many ways. It can be simplified or expanded in any way you like. For example:

f(y) = 5 * 4 + 10
Can be written as:
f(y) = (2+3) * (2+2) + 9 + 1

Or:
i,j= 0;
inc(y) = x+1
f(y) = inc(inc(inc(inc(inc(i))))) * inc(inc(inc(inc(inc(j))))) + inc(j+i)

Additionally, a Boolean expression (x OR y):

y	x		
	y v x	0	1
	0	0	1
	1	1	1

Can be expressed as NOT(NOT(x) AND NOT(y)):

y	x		
	¬(¬y ^ ¬x)	0	1
	0	0	1
	1	1	1

1. ADD <register/memory>, <value>

The ADD instruction is simple it adds the second operant to the first. Cases:

- You have a register that holds a value X (ADD EAX,<value>)
- You have a register that points to a memory offset that holds a value X (ADD [EAX],<value>)
- You have a direct memory offset that holds a value X (ADD [<offset>],<value>

In elementary math that is presented as:

y = y + x

This is equal to:

y = y - (-x)

In low level programming that can be written as:

SUB y,-x

This can be replaced with:

```
SUB <register/memory>, -<value>
```

Or by using LEA:

```
LEA <register>, DWORD [<register>+<value>]
```

(Note that: you cannot directly reference to a memory offset using the LEA instruction. Although, if needed you could load the memory value to a register, apply this instruction and finally put it back. Note that DWORD could also be BYTE or WORD)

2. SUB <register/[memory]>, <value>

Similarly, the ADD instruction, the same cases apply. SUB subtracts the second operant from the first. Therefore functions like:

y = y – x

Can be expressed as:

y = y + (-x)

This can be replaced with:

```
ADD <register/memory>, -<value>
```

Using LEA:

```
LEA <register>, DWORD[<register>-<value>]
```

(Note that: you cannot directly reference to a memory offset using the LEA instruction. Although, if needed you could load the memory value to a register, apply this instruction and write it back)

3. ADD <register,[memory]>,1

Adding the value one to a register or a memory location can be also expressed by NEG(NOT(value)). Note that this affects CF and AF flags (Carry Flag, Auxiliary Flag)

This can be replaced with:

```
NOT <register/[memory]>
NEG <register/[memory]>
```

Using LEA:

```
LEA <register>, DWORD [<register>+1]
```

4. SUB <register/[memory]>,1

Subtracting from a register or a value in memory the value 1

This can be replaced with:

```
NEG <register/[memory]>
NOT <register/[memory]>
```

Using LEA:

```
LEA <register>, DWORD [<register>-1]
```

5. MOV <register>, 0

Changing the value of a register to zero (0). Note that this could be combined with section 5 of Stack Based obfuscations.

This can be replaced with:

```
AND <register>,0
```

Or:

```
SHL <register>, 30
SHL <register>, 30
```

(Note that: SHL can affect Z,S and O flags)

Or:

```
XOR <register>,<register>
```

Or:

```
SUB <register>,<register>
```

Or:

```
LEA <register>, DWORD[0]
```

Additionally, changing the value of a(n) 16bit or 8bit register could also be achieved by:

16bit (for AX):

```
AND <register>, FFFF0000
```

8bit (for AL):

```
AND <register>, FFFFFF00
```

6. NOP

No operation, no impact on the CPU and memory. It only changes the EIP. NOP is equivalent to the instruction XCHG (E)AX,(E)AX.

This can be replaced with:

```
AND <register/[memory]>,FFFFFFFF (-1)
```

(3 byte with register, 7 byte with memory)

```
SHL <register>, 0
```

(3 byte)

```
SHR <register>, 0
```

(3 byte)

```
MOV <register>, <register>
```

(2 byte)

```
XOR <register/memory>, 0
```

(3 byte with register, 7 byte with memory (need write permissions), also affects P, A, Z flags)

```
ADD <register/[memory]>, 0
```

(3 byte with register, 7 byte with memory)

```
SUB <register/[memory],0
```

(3 byte with register, 7 byte with memory)

```
OR <register>,<register>
```

(3 byte with register, 7 byte with memory (need write permissions), also affects P, A, Z flags)

```
XCHG <register>
```

(2 byte if any register other than EAX is used, 1 byte otherwise)

```
LEA <register>,[<register>]
```

(2 byte)

```
FNOP
```

(2 byte affects C0, C1, C2, C3)

Multi-byte sequence of NOP instructions (available on processors with model encoding)

Length	Assembly	Byte Sequence
1 byte	NOP	90h
2 bytes	66 NOP	66 90h
3 bytes	NOP DWORD ptr [EAX]	0F 1F 00h
4 bytes	NOP DWORD ptr [EAX + 00H]	0F 1F 40 00h
5 bytes	NOP DWORD ptr [EAX + EAX*1 + 00H]	0F 1F 44 00 00h
6 bytes	66 NOP DWORD ptr [EAX + EAX*1 + 00H]	66 0F 1F 44 00 00h
7 bytes	NOP DWORD ptr [EAX + 00000000H]	0F 1F 80 00 00 00 00h
8 bytes	NOP DWORD ptr [EAX + EAX*1 + 00000000H]	0F 1F 84 00 00 00 00 00h
9 bytes	66 NOP DWORD ptr [EAX + EAX*1 + 00000000H]	66 0F 1F 84 00 00 00 00 00h

7. NOT<register/[memory]>

This can be replaced with:

```
XOR <register/[memory]>,-1
```

Or:

```
NEG <register/[memory]>
SUB <register/[memory]>,1
```

8. CMP <register>,0

This can be replaced with:

```
OR <register>,<register>
```

```
AND <register>,<register>
```

```
TEST <register>,<register>
```

9. NEG <register/[memory]>

This can be replaced with:

```
NOT <register/[memory]>
ADD <register/[memory]>,1
```

10. MOV <register>,<value>

This can be replaced with:

```
LEA <register>,[<value>]
```

Additional Obfuscations

SAHF/LAHF

These instructions are responsible for loading and storing the CPU flags (EFLAGS) SF, ZF, PF, AF, and CF to the AH register and storing them back.

These instructions execute as described above in compatibility mode and legacy mode. They are valid in 64-bit mode only if CPUID.80000001H:ECX.LAHF-SAHF[bit 0] = 1.

LAHF

Loads flags to the corresponding bits in AH register as shown below:

SF	ZF	**	AF	**	PF	**	CF
0	0	0	0	0	0	1	0

*** Reserved bits*

SAHF

Stores the contents of AH register to flags.

An example code for using SAHF and LAHF instructions

```
XOR EAX, EAX
XOR EBX, EBX
MOV AL, 1        ; Set EAX = 00000001 and EBX = 00000000
CMP EAX, EBX     ; Compare EAX with EBX => PF = 0 ZF = 0
LAHF             ; Store Flags in AH, EAX = 00000201
TEST EBX, EBX    ; PF = 1 ZF = 1
SAHF             ; Load Flags from AH => PF = 0 ZF = 0
JNZ <address>    ; Jump if Not Zero (if ZF is equal to 0)
```

Polymorphism and Self Modifying Code

Polymorphisism techniques can be used to encrypt/decrypt code. You can either use a stub to decrypt the whole code and execute it or decrypt sections of the code needed for execution and then discard them (a type of Self Modifying Code). Polymorphisism and Self Modifying Code do not fall under the scope of this paper.

Example Software

Program Name: getWAN
Md5sum: 1b73530132dd7f66fea5f29beb7a6c60
Compiler: MASM (WinAsm)

Program Analysis

Source Code

getWAN.asm

```
.486
.model   flat, stdcall
option   casemap :none

include          getWAN.inc

.code
start:
         invoke   GetModuleHandle, NULL
         mov      hInstance, eax
         invoke   DialogBoxParam, hInstance, 101, 0, ADDR DlgProc, 0
         invoke   ExitProcess, eax

DlgProc  proc     hWin     :DWORD,
                  uMsg     :DWORD,
                  wParam   :DWORD,
                  lParam   :DWORD
         .if      uMsg == WM_COMMAND
                  .if      wParam == GET
                           invoke GetWANIP,addr Weburl
                           invoke SetDlgItemText,hWin,RESULT,addr html
      .elseif     wParam == EXIT
                           invoke EndDialog,hWin,0
                  .endif
         .elseif  uMsg == WM_CLOSE
                  invoke   EndDialog,hWin,0
         .endif
         xor      eax,eax
         ret
DlgProc  endp

GetWANIP PROC url:DWORD
         local hInternet:DWORD
         local hURL:DWORD
         local hFile:DWORD
         local len:DWORD

         invoke InternetOpen,addr ipszAgent,INTERNET_OPEN_TYPE_PRECONFIG,0,0,0
         mov hInternet,eax

         invoke InternetOpenUrl,hInternet,url,0,0,0,0
```

```
        mov hURL,eax

        invoke InternetReadFile,hURL,addr html,sizeof html,addr len

        invoke InternetCloseHandle,hURL
        .if eax==0
                invoke InternetCloseHandle,hInternet
                Ret
        .endif
        Ret
GetWANIP EndP

end start
```

getWAN.inc

```
include         windows.inc
include         user32.inc
include         kernel32.inc
include         wsock32.inc
include         wininet.inc
;-------------------------------------------------------------------------------------------------------------------
includelib      user32.lib
includelib      kernel32.lib
includelib      wsock32.lib
includelib      wininet.lib
;-------------------------------------------------------------------------------------------------------------------
DlgProc         PROTO :DWORD,:DWORD,:DWORD,:DWORD
GetWANIP        PROTO :DWORD
;-------------------------------------------------------------------------------------------------------------------
GET             equ     1001
EXIT            equ     1002
RESULT          equ     1003
;-------------------------------------------------------------------------------------------------------------------
.data
        Weburl                  db              "http://www.whatismyip.com/automation/n09230945.asp",0
        ipszAgent               db              "Mozilla/6.9",0
.data?
        hInstance               dd              ?
        buffer                  db              256 dup(?)
        html                    db              256 dup(?)
;-------------------------------------------------------------------------------------------------------------------
```

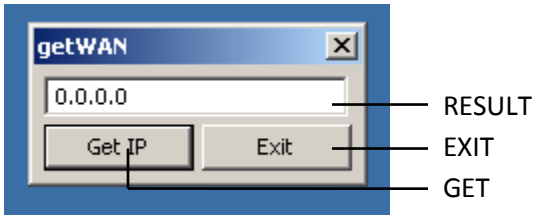
getWAN.rc

```
;This Resource Script was generated by WinAsm Studio.

#define GET     1001
#define EXIT    1002
#define RESULT  1003


101             DIALOGEX 0,0,109,35
CAPTION         "getWAN"
FONT            8,"Tahoma"
STYLE           0x80c80880
EXSTYLE         0x00000000
BEGIN
        CONTROL "Get IP",GET,"Button",0x10000001,3,18,50,14,0x00000000
        CONTROL "Exit",EXIT,"Button",0x10000000,56,18,50,14,0x00000000
        CONTROL "",RESULT,"Edit",0x10000080,3,3,101,12,0x00000200
END
```

User Interface



Assembled Code

```
00401000  >/$  6A 00              PUSH 0                                   ; /pModule = NULL
00401002  |.  E8 FB000000        CALL <JMP.&kernel32.GetModuleHandleA>    ; \GetModuleHandleA
00401007  |.  A3 40304000        MOV DWORD PTR DS:[403040],EAX
0040100C  |.  6A 00              PUSH 0                                   ; /lParam = NULL
0040100E  |.  68 28104000        PUSH getWAN.00401028                     ; |DlgProc = getWAN.00401028
00401013  |.  6A 00              PUSH 0                                   ; |hOwner = NULL
00401015  |.  6A 65              PUSH 65                                  ; |pTemplate = 65
00401017  |.  FF35 40304000      PUSH DWORD PTR DS:[403040]               ; |hInst = NULL
0040101D  |.  E8 C8000000        CALL <JMP.&user32.DialogBoxParamA>       ; \DialogBoxParamA
00401022  |.  50                 PUSH EAX                                 ; /ExitCode
00401023  \.  E8 D4000000        CALL <JMP.&kernel32.ExitProcess>         ; \ExitProcess
00401028  /.  55                 PUSH EBP
00401029  |.  8BEC               MOV EBP,ESP
0040102B  |.  817D 0C 11010>     CMP DWORD PTR SS:[EBP+C],111
00401032  |.  75 3C              JNZ SHORT getWAN.00401070
00401034  |.  817D 10 E9030>     CMP DWORD PTR SS:[EBP+10],3E9
0040103B  |.  75 1E              JNZ SHORT getWAN.0040105B
0040103D  |.  68 00304000        PUSH getWAN.00403000                     ; /Arg1 = 00403000 ASCII
                                                                          "http://www.whatismyip.com/
                                                                          automation/n09230945.asp"
```

```
00401042  |.   E8 3F000000     CALL getWAN.00401086                    ; \getWAN.00401086
00401047  |.   68 44314000     PUSH getWAN.00403144                    ; /Text = ""
0040104C  |.   68 EB030000     PUSH 3EB                                ; |ControlID = 3EB (1003.)
00401051  |.   FF75 08         PUSH DWORD PTR SS:[EBP+8]               ; |hWnd
00401054  |.   E8 9D000000     CALL <JMP.&user32.SetDlgItemTextA>      ; \SetDlgItemTextA
00401059  |.   EB 25           JMP SHORT getWAN.00401080
0040105B  |>  817D 10 EA030>  CMP DWORD PTR SS:[EBP+10],3EA
00401062  |.   75 1C           JNZ SHORT getWAN.00401080
00401064  |.   6A 00           PUSH 0                                  ; /Result = 0
00401066  |.   FF75 08         PUSH DWORD PTR SS:[EBP+8]               ; |hWnd
00401069  |.   E8 82000000     CALL <JMP.&user32.EndDialog>            ; \EndDialog
0040106E  |.   EB 10           JMP SHORT getWAN.00401080
00401070  |>  837D 0C 10      CMP DWORD PTR SS:[EBP+C],10
00401074  |.   75 0A           JNZ SHORT getWAN.00401080
00401076  |.   6A 00           PUSH 0                                  ; /Result = 0
00401078  |.   FF75 08         PUSH DWORD PTR SS:[EBP+8]               ; |hWnd
0040107B  |.   E8 70000000     CALL <JMP.&user32.EndDialog>            ; \EndDialog
00401080  |>  33C0             XOR EAX,EAX
00401082  |.   C9              LEAVE
00401083  \.   C2 1000         RETN 10
00401086  /$   55              PUSH EBP
00401087  |.   8BEC            MOV EBP,ESP
00401089  |.   83C4 F0         ADD ESP,-10
0040108C  |.   6A 00           PUSH 0
0040108E  |.   6A 00           PUSH 0
00401090  |.   6A 00           PUSH 0
00401092  |.   6A 00           PUSH 0
00401094  |.   68 33304000     PUSH getWAN.00403033                    ;  ASCII "Mozilla/6.9"
00401099  |.   E8 70000000     CALL <JMP.&wininet.InternetOpenA>
0040109E  |.   8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
004010A1  |.   6A 00           PUSH 0
004010A3  |.   6A 00           PUSH 0
004010A5  |.   6A 00           PUSH 0
004010A7  |.   6A 00           PUSH 0
004010A9  |.   FF75 08         PUSH DWORD PTR SS:[EBP+8]
004010AC  |.   FF75 FC         PUSH DWORD PTR SS:[EBP-4]
004010AF  |.   E8 60000000     CALL <JMP.&wininet.InternetOpenUrlA>
004010B4  |.   8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
004010B7  |.   8D45 F0         LEA EAX,DWORD PTR SS:[EBP-10]
004010BA  |.   50              PUSH EAX
004010BB  |.   68 00010000     PUSH 100
004010C0  |.   68 44314000     PUSH getWAN.00403144
004010C5  |.   FF75 F8         PUSH DWORD PTR SS:[EBP-8]
004010C8  |.   E8 4D000000     CALL <JMP.&wininet.InternetReadFile>
004010CD  |.   FF75 F8         PUSH DWORD PTR SS:[EBP-8]
004010D0  |.   E8 33000000     CALL <JMP.&wininet.InternetCloseHandle>
004010D5  |.   0BC0            OR EAX,EAX
004010D7  |.   75 0C           JNZ SHORT getWAN.004010E5
004010D9  |.   FF75 FC         PUSH DWORD PTR SS:[EBP-4]
004010DC  |.   E8 27000000     CALL <JMP.&wininet.InternetCloseHandle>
004010E1  |.   C9              LEAVE
004010E2  |.   C2 0400         RETN 4
```


```
004010E5  |>     C9              LEAVE
004010E6  \.     C2 0400         RETN 4
004010E9         CC              INT3
004010EA   $- FF25 14204000 JMP DWORD PTR DS:[<&user32.DialogBoxPara>;  user32.DialogBoxParamA
004010F0   $- FF25 10204000 JMP DWORD PTR DS:[<&user32.EndDialog>]   ;  user32.EndDialog
004010F6   $- FF25 0C204000 JMP DWORD PTR DS:[<&user32.SetDlgItemTex>;  user32.SetDlgItemTextA
004010FC   .- FF25 04204000 JMP DWORD PTR DS:[<&kernel32.ExitProcess>;  kernel32.ExitProcess
00401102   $- FF25 00204000 JMP DWORD PTR DS:[<&kernel32.GetModuleHa>;  kernel32.GetModuleHandleA
00401108   $- FF25 28204000 JMP DWORD PTR DS:[<&wininet.InternetClos>;  wininet.InternetCloseHandle
0040110E   $- FF25 1C204000 JMP DWORD PTR DS:[<&wininet.InternetOpen>;  wininet.InternetOpenA
00401114   $- FF25 20204000 JMP DWORD PTR DS:[<&wininet.InternetOpen>;  wininet.InternetOpenUrlA
0040111A   $- FF25 24204000 JMP DWORD PTR DS:[<&wininet.InternetRead>;  wininet.InternetReadFile
```

General Obfuscation approach

The following assembly code is an example of how you could obfuscate the code.

```
00401000 >    C74424 FC 000>     MOV DWORD PTR SS:[ESP-4],0
00401008      83EC 04           SUB ESP,4
0040100B      E8 00000000       CALL getWAN_o.00401010
00401010   $  8F05 40304000     POP DWORD PTR DS:[403040]
00401016   .  832D 40304000>    SUB DWORD PTR DS:[403040],-18
0040101D   .  FF35 40304000     PUSH DWORD PTR DS:[403040]
00401023   .  E9 DA000000       JMP <JMP.&kernel32.GetModuleHandleA>
00401028   .  A3 40304000       MOV DWORD PTR DS:[403040],EAX
0040102D   .  C74424 FC 000>    MOV DWORD PTR SS:[ESP-4],0
00401035   .  83EC 04           SUB ESP,4
00401038   .  68 79104000       PUSH getWAN_o.00401079
0040103D   .  6A 00             PUSH 0
0040103F   .  C74424 FC 650>    MOV DWORD PTR SS:[ESP-4],65
00401047   .  83EC 04           SUB ESP,4
0040104A   .  FF35 40304000     PUSH DWORD PTR DS:[403040]
00401050   .  E8 00000000       CALL getWAN_o.00401055
00401055   $  8F05 50304000     POP DWORD PTR DS:[403050]
0040105B   .  832D 50304000>    SUB DWORD PTR DS:[403050],-18
00401062   .  FF35 50304000     PUSH DWORD PTR DS:[403050]
00401068   .  E9 7D000000       JMP <JMP.&user32.DialogBoxParamA>
0040106D   .  894424 FC         MOV DWORD PTR SS:[ESP-4],EAX
00401071   .  83EC 04           SUB ESP,4
00401074   .  E9 83000000       JMP <JMP.&kernel32.ExitProcess>
00401079   .  55                PUSH EBP
0040107A   .  54                PUSH ESP
0040107B   .  5D                POP EBP
0040107C   .  817D 0C 11010>    CMP DWORD PTR SS:[EBP+C],111
00401083   .  75 51             JNZ SHORT getWAN_o.004010D6
00401085   .  817D 10 E9030>    CMP DWORD PTR SS:[EBP+10],3E9
0040108C   .  75 2C             JNZ SHORT getWAN_o.004010BA
0040108E   .  68 00304000       PUSH getWAN_o.00403000          ;  ASCII "http://www.whatismyi...."
00401093   .  B8 37114000       MOV EAX,getWAN_o.00401137
00401098   .  FFD0              CALL EAX
0040109A   .  68 44314000       PUSH getWAN_o.00403144
```

```
0040109F   .  C74424 FC EB0>     MOV DWORD PTR SS:[ESP-4],3EB
004010A7   .  83EC 04            SUB ESP,4
004010AA   .  FF75 08            PUSH DWORD PTR SS:[EBP+8]
004010AD   .  E8 00000000        CALL getWAN_o.004010B2
004010B2   $  830424 06          ADD DWORD PTR SS:[ESP],6
004010B6   .  EB 3E              JMP SHORT <JMP.&user32.SetDlgItemTextA>
004010B8   .  EB 6C              JMP SHORT getWAN_o.00401126
004010BA   >  B8 EA030000        MOV EAX,3EA
004010BF   .  3945 10            CMP DWORD PTR SS:[EBP+10],EAX
004010C2   .  75 62              JNZ SHORT getWAN_o.00401126
004010C4   .  6A 00              PUSH 0
004010C6   .  FF75 08            PUSH DWORD PTR SS:[EBP+8]
004010C9   >  E8 00000000        CALL getWAN_o.004010CE
004010CE   $  830424 07          ADD DWORD PTR SS:[ESP],7
004010D2   .  EB 1C              JMP SHORT <JMP.&user32.EndDialog>
004010D4   .  EB 50              JMP SHORT getWAN_o.00401126
004010D6   >  05 26FCFFFF        ADD EAX,-3DA
004010DB   .  837D 0C 10         CMP DWORD PTR SS:[EBP+C],10
004010DF   .  75 45              JNZ SHORT getWAN_o.00401126
004010E1   .  6A 00              PUSH 0
004010E3   .  EB 3C              JMP SHORT getWAN_o.00401121
004010E5      CC                 INT3
004010E6      CC                 INT3
004010E7      CC                 INT3
004010E8      CC                 INT3
004010E9      CC                 INT3
004010EA   >- FF25 14204000 JMP DWORD PTR DS:[<&user32.DialogBoxPara>;  user32.DialogBoxParamA
004010F0   >- FF25 10204000 JMP DWORD PTR DS:[<&user32.EndDialog>]   ;  user32.EndDialog
004010F6   >- FF25 0C204000 JMP DWORD PTR DS:[<&user32.SetDlgItemTex>;  user32.SetDlgItemTextA
004010FC   >- FF25 04204000 JMP DWORD PTR DS:[<&kernel32.ExitProcess>;  kernel32.ExitProcess
00401102   >- FF25 00204000 JMP DWORD PTR DS:[<&kernel32.GetModuleHa>;  kernel32.GetModuleHandleA
00401108   $- FF25 28204000 JMP DWORD PTR DS:[<&wininet.InternetClos>;  wininet.InternetCloseHandle
0040110E   $- FF25 1C204000 JMP DWORD PTR DS:[<&wininet.InternetOpen>;  wininet.InternetOpenA
00401114   $- FF25 20204000 JMP DWORD PTR DS:[<&wininet.InternetOpen>;  wininet.InternetOpenUrlA
0040111A   $- FF25 24204000 JMP DWORD PTR DS:[<&wininet.InternetRead>;  wininet.InternetReadFile
00401120      00             DB 00
00401121   >  FF75 08         PUSH DWORD PTR SS:[EBP+8]
00401124   .^ EB A3           JMP SHORT getWAN_o.004010C9
00401126   >  6A 00           PUSH 0
00401128      660F1F84000000> NOP
00401131      58             POP EAX
00401132      C9             LEAVE
00401133   .  C2 1000         RETN 10
00401136      00             DB 00
00401137   /.  55             PUSH EBP
00401138   |.  8BEC           MOV EBP,ESP
0040113A   |.  83C4 F0        ADD ESP,-10
0040113D   |.  6A 00          PUSH 0
0040113F   |.  6A 00          PUSH 0
00401141   |.  6A 00          PUSH 0
00401143   |.  6A 00          PUSH 0
00401145   |.  68 33304000    PUSH getWAN_o.00403033          ;  ASCII "Mozilla/6.9"
```

```
0040114A  |.  E8 BFFFFFFF       CALL <JMP.&wininet.InternetOpenA>
0040114F  |.  8945 FC           MOV DWORD PTR SS:[EBP-4],EAX
00401152  |.  6A 00             PUSH 0
00401154  |.  6A 00             PUSH 0
00401156  |.  6A 00             PUSH 0
00401158  |.  6A 00             PUSH 0
0040115A  |.  FF75 08           PUSH DWORD PTR SS:[EBP+8]
0040115D  |.  FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00401160  |.  E8 AFFFFFFF       CALL <JMP.&wininet.InternetOpenUrlA>
00401165  |.  8945 F8           MOV DWORD PTR SS:[EBP-8],EAX
00401168  |.  8D45 F0           LEA EAX,DWORD PTR SS:[EBP-10]
0040116B  |.  50                PUSH EAX
0040116C  |.  68 00010000       PUSH 100
00401171  |.  68 44314000       PUSH getWAN_o.00403144
00401176  |.  FF75 F8           PUSH DWORD PTR SS:[EBP-8]
00401179  |.  E8 9CFFFFFF       CALL <JMP.&wininet.InternetReadFile>
0040117E  |.  FF75 F8           PUSH DWORD PTR SS:[EBP-8]
00401181  |.  E8 82FFFFFF       CALL <JMP.&wininet.InternetCloseHandle>
00401186  |.  0BC0              OR EAX,EAX
00401188  |.  75 0C             JNZ SHORT getWAN_o.00401196
0040118A  |.  FF75 FC           PUSH DWORD PTR SS:[EBP-4]
0040118D  |.  E8 76FFFFFF       CALL <JMP.&wininet.InternetCloseHandle>
00401192  |.  C9                LEAVE
00401193  |.  C2 0400           RETN 4
00401196  |>  C9                LEAVE
00401197  \.  C2 0400           RETN 4
```

Obfuscation Index Table

Instruction	Equal Instructions

PUSH <value/register>	MOV DWORD PTR SS:[ESP-4],<value/register>
	SUB ESP,4
	OR
	MOV <register/[memory]>,<value/register>
	PUSH <register/[memory]>

POP <register/[memory]>	MOV <register>, DWORD [ESP]
	ADD ESP,4
	OR
	POP <[memory]>
	MOV <register>,<[memory]>

CALL <address>	PUSH EIP + bytes to next instruction JMP <address> OR MOV <[memory]/register>,<address> CALL <[memory]/register>
JMP <address>	PUSH <address> RETN OR CALL <address> At address: POP <register> or ADD ESP,4
RETN	POP EAX JMP EAX
SUB <register/memory>,<value>	ADD <register>, -<value> OR LEA <register>, DWORD [<register>-<value>]
ADD <register/memory>,<value>	SUB <register>,-<value> OR LEA <register>, DWORD [<register>+<value>]
PUSH <x> POP <register>	MOV <register>,<x>
ADD <register/[memory]>,1	NOT <register/[memory]> NEG <register/[memory]> (affects C and A flags) LEA <register>,DWORD[<register>+1]
SUB <register/[memory]>,1	NEG <register/[memory]> NOT <register/[memory]> OR LEA <register>,DWORD[<register>-1]
***MOV <register>,0	AND <register>,0 OR SHL <register>,30 SHL <register>,30

	(note that this affects the Z,S and O flags) OR PUSH 0 POP <register> OR XOR <register>,<register> OR SUB <register>,<register> OR LEA <register>, DWORD [0]
MOV <16bit register>,0	AND <register>,FFFF0000 AND <16bit register>, 0000

NOP	AND <register/[memory]>,FFFFFFFF (-1)
	OR
	SHL <register>,0 (logical shift left)
	OR
	SHR <register>,0 (logical shift right)
	OR
	MOV <register>,<register>
	OR
	XOR <register/[memory]>,0
	OR
	ADD <register/[memory]>,0
	OR
	SUB <register/[memory]>,0
	OR
	OR <register/[memory]>,0
	OR
	PUSH <register/[memory]>
	POP <register/[memory]>
	OR
	XCHG <register>
	OR
	FNOP
	OR
	LEA <register>, [<register>]
	OR
	FNOP

NOT <register/[memory]>	XOR <register/[memory]>,-1 OR NEG <register/[memory]> SUB <register/[memory]>,1
MOV <register a>,<register b>	PUSH <register a> POP <register b> OR

	MOV DWORD [<[memory] offset>],<register a> MOV <register b>, DWORD [<[memory] offset>] OR LEA <register a>, DWORD [<register b>] OR MOV <[memory]>,<register a> MOV <register b>,<[memory]>
CMP <register>,0	OR <register>,<register> OR AND <register>,<register> OR TEST <register>,<register>
**MOV <register>,<value>	LEA <register>,[<value>] OR PUSH <value> POP <register>
MOV <[memory]>,<value/register>	PUSH <value/register> POP <[memory]>
MOV <[memory] a>,<[memory] b>* (NO such instruction)	PUSH <[memory] b> POP <[memory] a>
NEG <register/[memory]>	NOT <register/[memory]> ADD <register/[memory]>,1

Greetings

I would like to say thank you to the people that helped me writing this paper. Special thanks fly out to Maria-Lena Demetriou for correcting and pointing out some grammatical errors :)