RIVERSIDE RESEARCH INSTITUTE

Deobfuscator:

An Automated Approach to the Identification and Removal of Code Obfuscation

Eric Laspe, Reverse Engineer

Jason Raber, Lead Reverse Engineer

Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

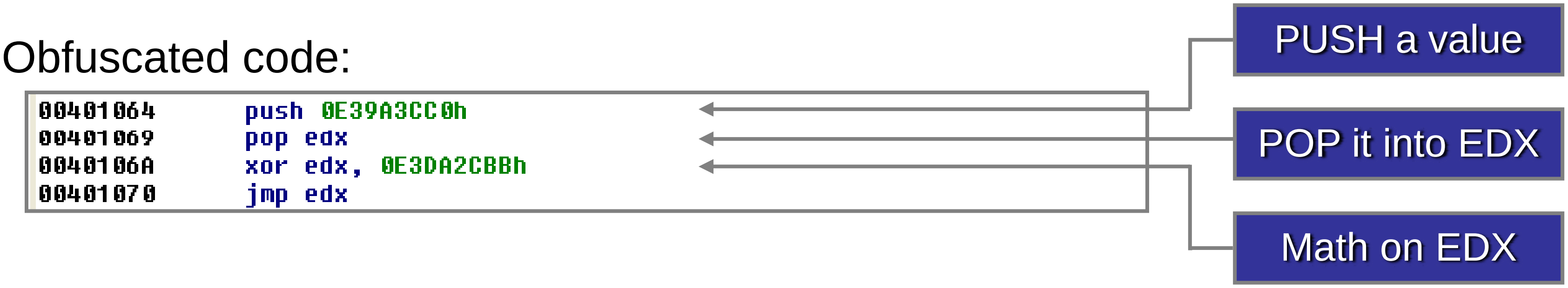
The Problem: Obfuscated Code

- Malware authors use code obfuscation techniques to hide their malicious code
- Obfuscation costs reverse engineers time:
  - Complicates instruction sequences
  - Disrupts control flow
  - Makes algorithms difficult to understand
- Manual obfuscation removal is a tedious and error-prone process

Example: PUSH_POP_MATH

PUSH an immediate, then POP into a register and do some math on it

Obfuscated code:



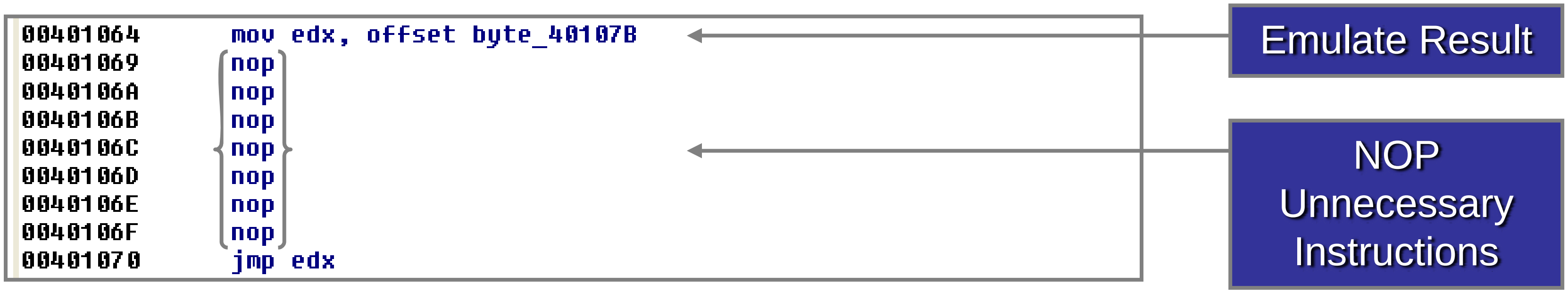
```
00401064        push 0E39A3CC0h
00401069        pop edx
0040106A        xor edx, 0E3DA2CBBh
00401070        jmp edx
```

PUSH a value

POP it into EDX

Math on EDX

Resolves to:



```
00401064        mov edx, offset byte_40107B
00401069        nop
0040106A        nop
0040106B        nop
0040106C        nop
0040106D        nop
0040106E        nop
0040106F        nop
00401070        jmp edx
```

Emulate Result

NOP Unnecessary Instructions

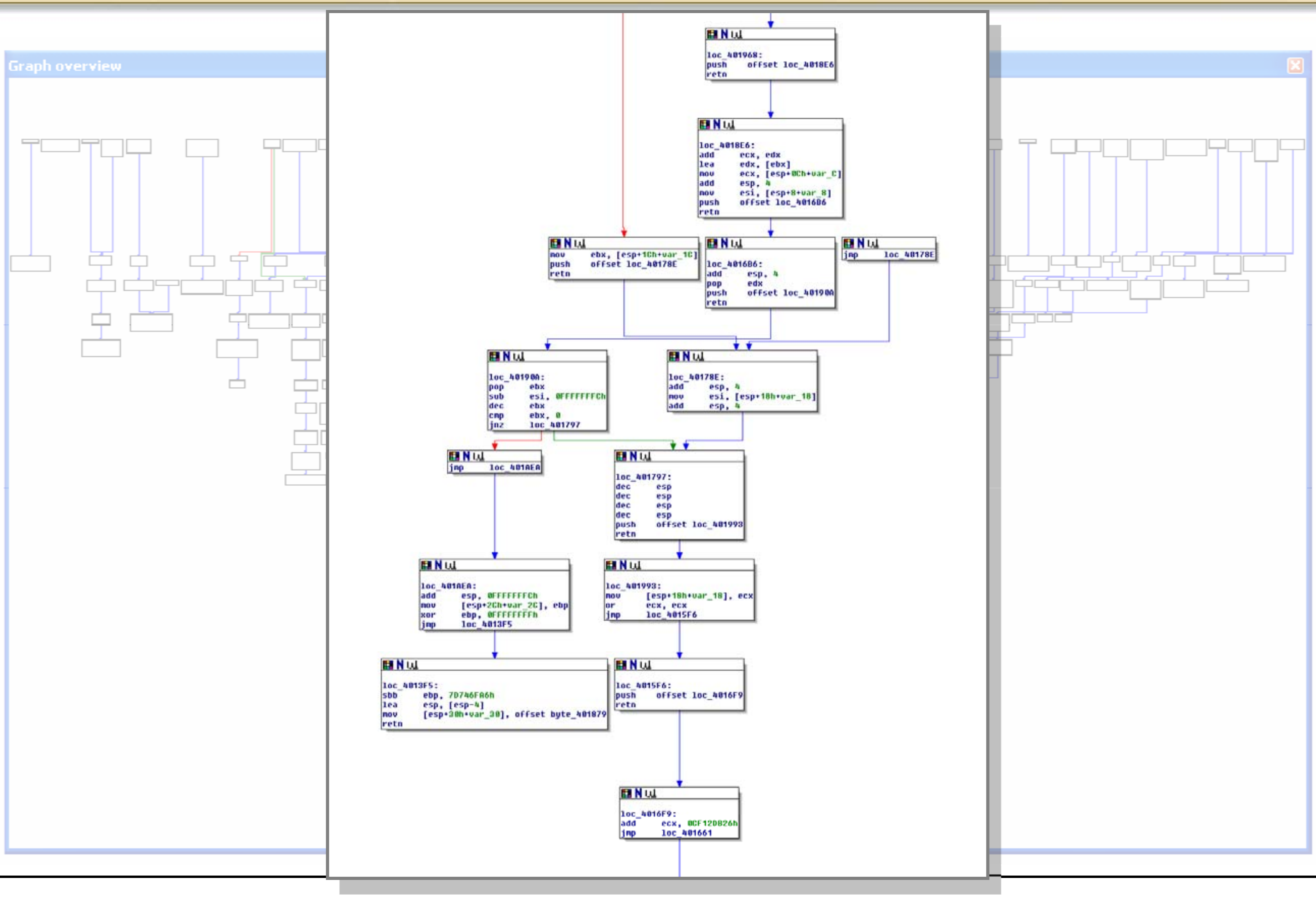
Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

Malware Example: RustockB

- Good malware example that implemented obfuscation patterns to hide a decryption routine
- Many useless and confusing instructions
  - Push regs, math, pop regs
  - Pushes and pops in various obfuscated forms
- Control flow obscured
  - Mangled jumps
  - Unnecessary data cross-references

RustockB Control Flow



```
loc_401968:
push    offset loc_4018E6
retn
```

```
loc_4018E6:
add     ecx, edx
lea     edx, [ebx]
mov     ecx, [esp+0Ch+var_C]
add     esp, 4
mov     esi, [esp+8+var_8]
push    offset loc_4016D6
retn
```

```
mov     ebx, [esp+1Ch+var_1C]
push    offset loc_40178E
retn
```

```
loc_4016D6:
add     esp, 4
pop     edx
push    offset loc_40190A
retn
```

```
jmp     loc_40178E
```

```
loc_40190A:
pop     ebx
sub     esi, 0FFFFFFFCh
dec     ebx
cmp     ebx, 0
jnz     loc_401797
```

```
loc_40178E:
add     esp, 4
mov     esi, [esp+18h+var_18]
add     esp, 4
```

```
jmp     loc_401AEA
```

```
loc_401797:
dec     esp
dec     esp
dec     esp
dec     esp
push    offset loc_401993
retn
```

```
loc_401AEA:
add     esp, 0FFFFFFFCh
mov     [esp+2Ch+var_2C], ebp
xor     ebp, 0FFFFFFFFh
jmp     loc_4013F5
```

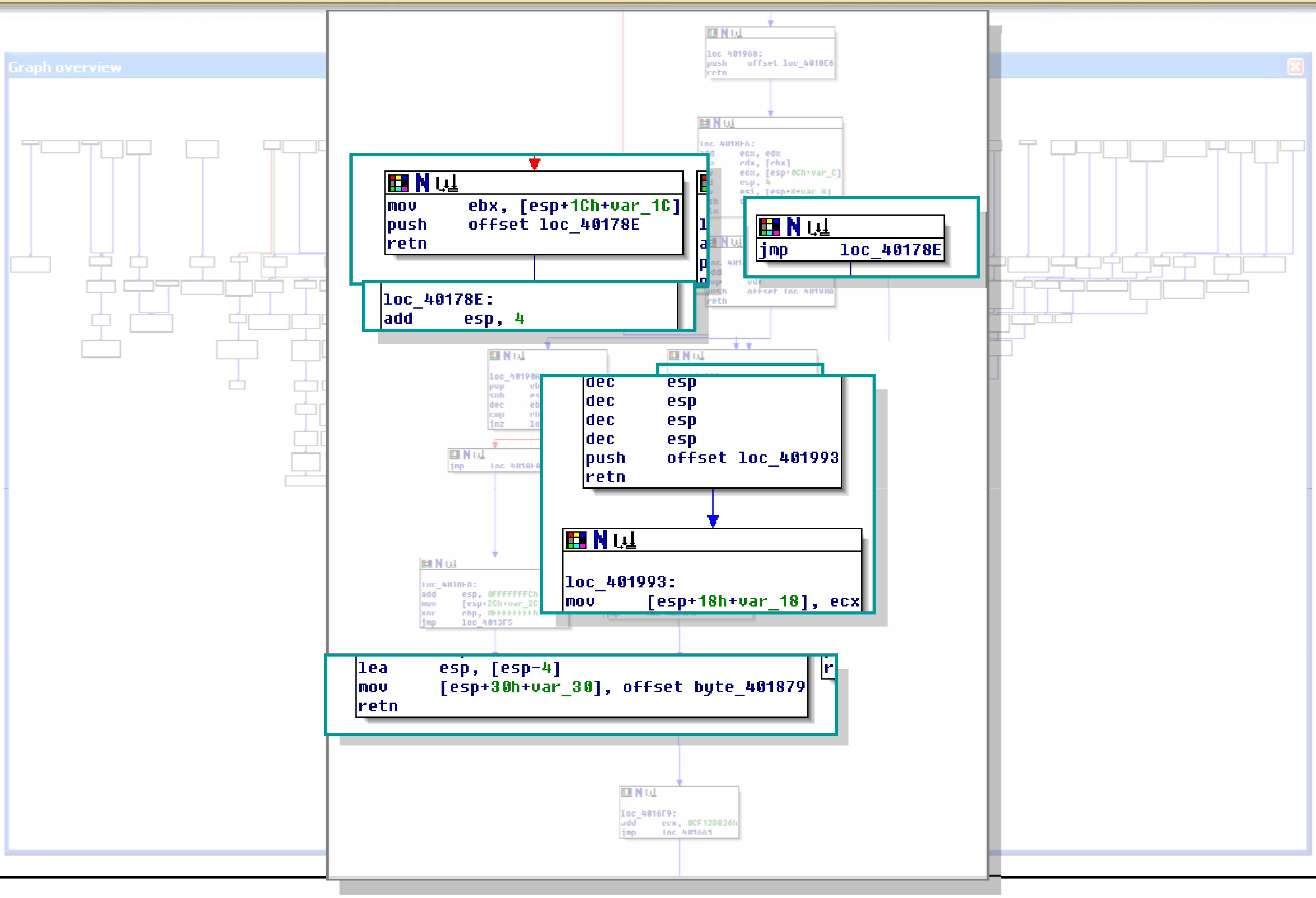
```
loc_401993:
mov     [esp+18h+var_18], ecx
or      ecx, ecx
jmp     loc_4015F6
```

```
loc_4013F5:
sbb     ebp, 7D746FA6h
lea     esp, [esp-4]
mov     [esp+30h+var_30], offset byte_401879
retn
```

```
loc_4015F6:
push    offset loc_4016F9
retn
```

```
loc_4016F9:
add     ecx, 0CF12D826h
jmp     loc_401661
```

RustockB Control Flow



```
mov     ebx, [esp+1Ch+var_1C]
push    offset loc_40178E
retn
```

```
loc_40178E:
add     esp, 4
```

```
jmp     loc_40178E
```

```
dec     esp
dec     esp
dec     esp
dec     esp
push    offset loc_401993
retn
```

```
loc_401993:
mov     [esp+18h+var_18], ecx
```

```
lea     esp, [esp-4]
mov     [esp+30h+var_30], offset byte_401879
retn
```

Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

The Solution:
The Deobfuscator IDA Pro Plug-in

- Combines instruction emulation and pattern recognition
- Determines proper code control flow
- Interprets and transforms instruction sequences to enhance code readability
- Uses a binary injector to make both static and dynamic analysis easier

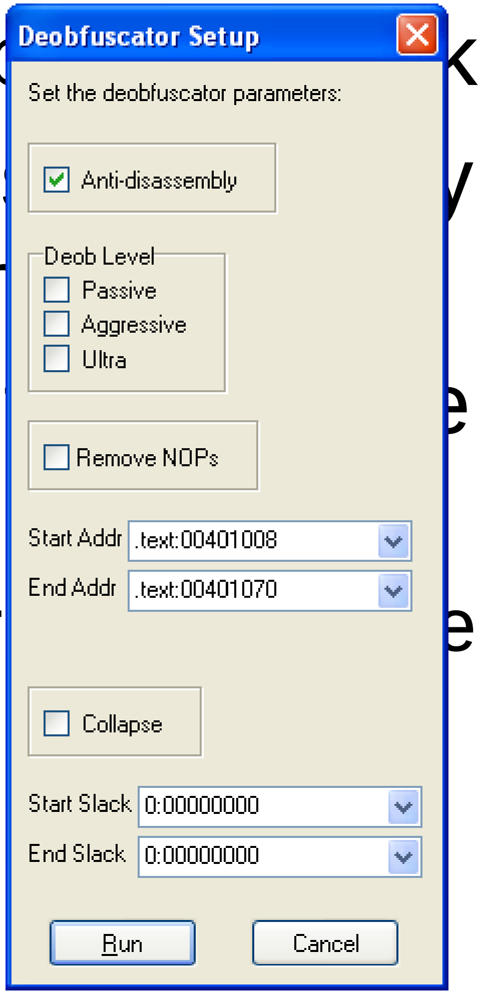
Modes of Operation

The plug-in has six modes:

- Anti-disassembly – replaces anti-disassembly with simplified code
- Passive – simple peep-hole rules
- Aggressive – uses aggressive assumptions about memory contents
- Ultra – more aggressive assumptions
- Remove NOPs – jumps over slack space
- NEW! Collapse – moves consecutive code blocks together to eliminate NOPs and JMPs

IDA Pro Integration

- Deobfuscator p ked with Alt-Z
- Uses structures y IDA Pro disassembly ar
- Depending on elected, it can:
  - Follow jumps
  - Track register e the stack



Deobfuscator Setup

Set the deobfuscator parameters:

☑ Anti-disassembly

Deob Level
□ Passive
□ Aggressive
□ Ultra

□ Remove NOPs

Start Addr .text:00401008

End Addr .text:00401070

□ Collapse

Start Slack 0:00000000

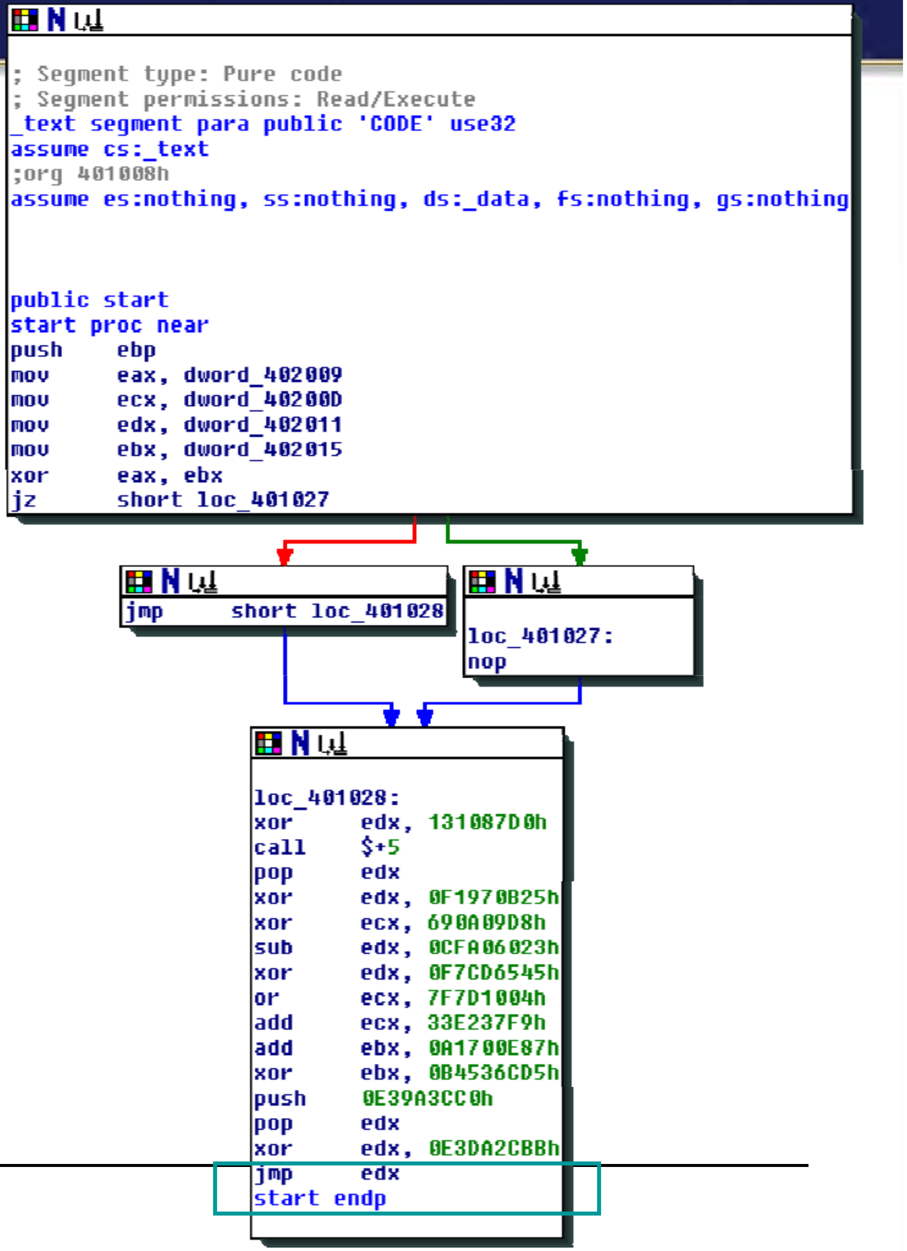
End Slack 0:00000000

Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

Demonstration

- Demo code protected with anti-disassembly code and obfuscation
- Note the obfuscated jump at the end of this graph
- Run iteratively, the Deobfuscator will remove obfuscation and improve code flow readability



```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use32
assume cs:_text
;org 401008h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing



public start
start proc near
push    ebp
mov     eax, dword_402009
mov     ecx, dword_40200D
mov     edx, dword_402011
mov     ebx, dword_402015
xor     eax, ebx
jz      short loc_401027
```

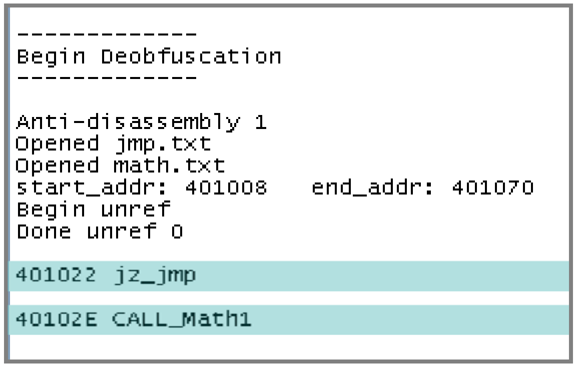
```
jmp     short loc_401028
```

```
loc_401027:
nop
```

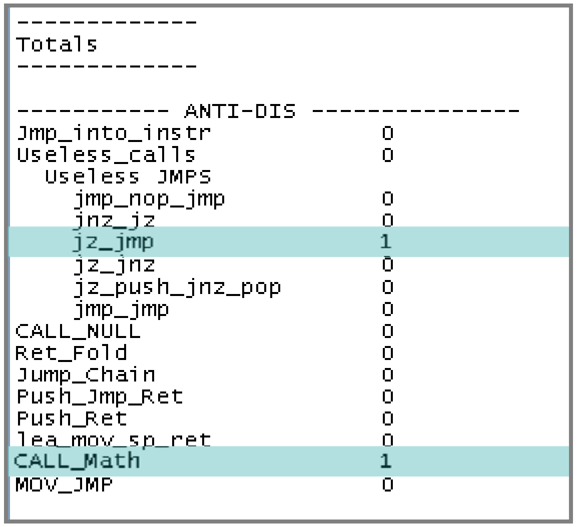
```
loc_401028:
xor     edx, 131087D0h
call    $+5
pop     edx
xor     edx, 0F1970B25h
xor     ecx, 690A09D8h
sub     edx, 0CFA06023h
xor     edx, 0F7CD6545h
or      ecx, 7F7D1004h
add     ecx, 33E237F9h
add     ebx, 0A1700E87h
xor     ebx, 0B4536CD5h
push    0E39A3CC0h
pop     edx
xor     edx, 0E3DA2CBBh
jmp     edx
start endp
```

Run 1 – Anti-Disassembly

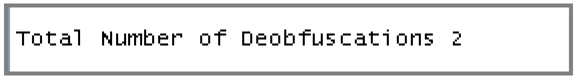
- Two matching patterns
  - JZ_JMP
  - CALL_MATH



```
-------------
Begin Deobfuscation
-------------

Anti-disassembly 1
Opened jmp.txt
Opened math.txt
start_addr: 401008    end_addr: 401070
Begin unref
Done unref 0

401022  jz_jmp

40102E  CALL_Math1
```



```
-------------
Totals
-------------

----------- ANTI-DIS ---------------
Jmp_into_instr              0
Useless_calls               0
  Useless JMPS
    jmp_nop_jmp             0
    jnz_jz                  0
    jz_jmp                  1
    jz_jnz                  0
    jz_push_jnz_pop         0
    jmp_jmp                 0
CALL_NULL                   0
Ret_Fold                    0
Jump_Chain                  0
Push_Jmp_Ret                0
Push_Ret                    0
lea_mov_sp_ret              0
CALL_Math                   1
MOV_JMP                     0
```

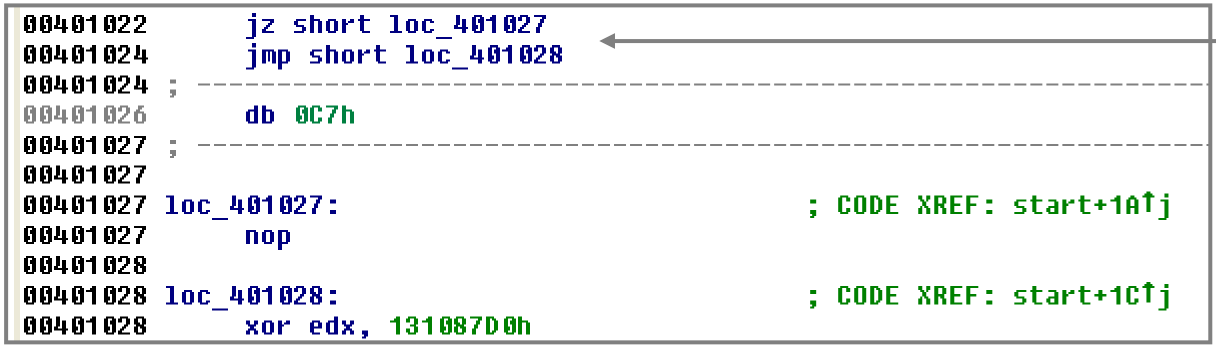


```
Total Number of Deobfuscations 2
```

Pattern: *JZ_JMP*

Two useless jumps

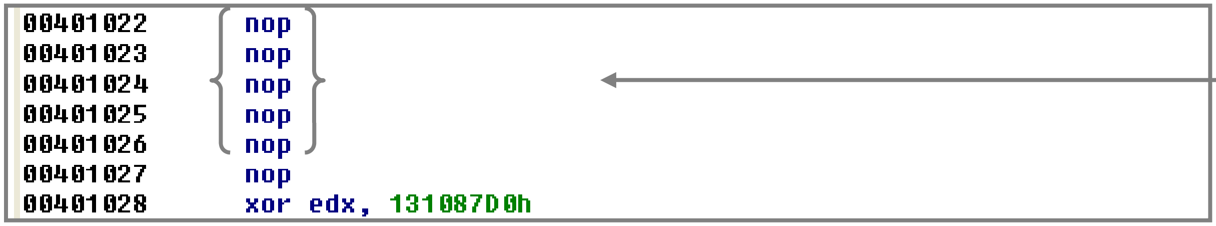
Before Deobfuscation:



```
00401022         jz short loc_401027
00401024         jmp short loc_401028
00401024 ; ---------------------------------------------------------------
00401026         db 0C7h
00401027 ; ---------------------------------------------------------------
00401027
00401027 loc_401027:                             ; CODE XREF: start+1A↑j
00401027         nop
00401028
00401028 loc_401028:                             ; CODE XREF: start+1C↑j
00401028         xor edx, 131087D0h
```

Useless Jumps

After Deobfuscation:



```
00401022         nop
00401023         nop
00401024         nop
00401025         nop
00401026         nop
00401027         nop
00401028         xor edx, 131087D0h
```

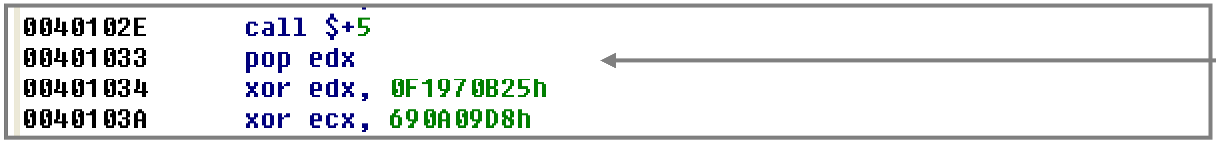
NOP'd Jumps

Pattern: *CALL_MATH*

EDX gets the return address of the CALL $5

Then, there is some math on EDX

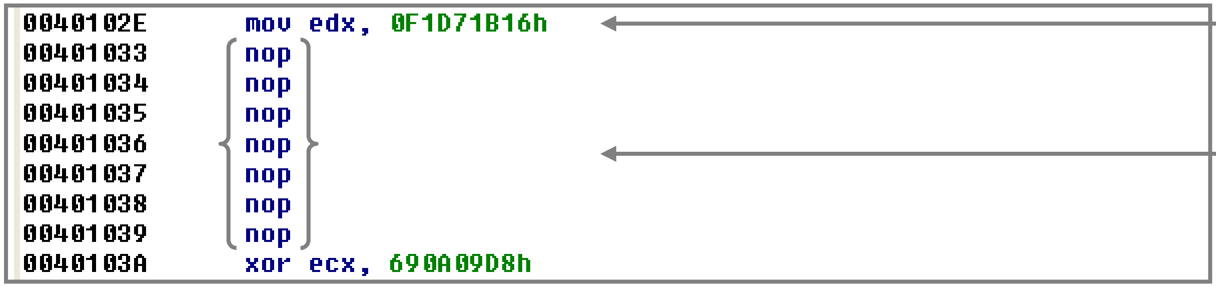
Before Deobfuscation:



```
0040102E        call $+5
00401033        pop edx
00401034        xor edx, 0F1970B25h
0040103A        xor ecx, 690A09D8h
```

EDX = 401033

After Deobfuscation:



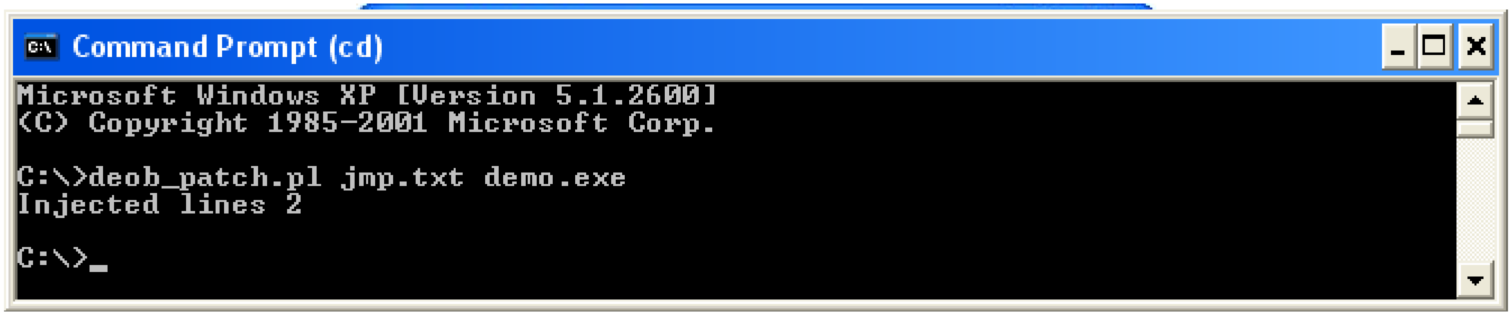
```
0040102E        mov edx, 0F1D71B16h
00401033        nop
00401034        nop
00401035        nop
00401036        nop
00401037        nop
00401038        nop
00401039        nop
0040103A        xor ecx, 690A09D8h
```

Emulated Result

NOP'd Pop & Math

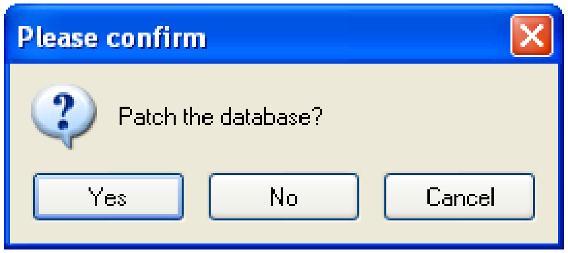
Output Injection

- A text file is generated by the Deobfuscator plug-in
- Then, we inject the binary with a PERL script



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>deob_patch.pl jmp.txt demo.exe
Injected lines 2

C:\>_
```

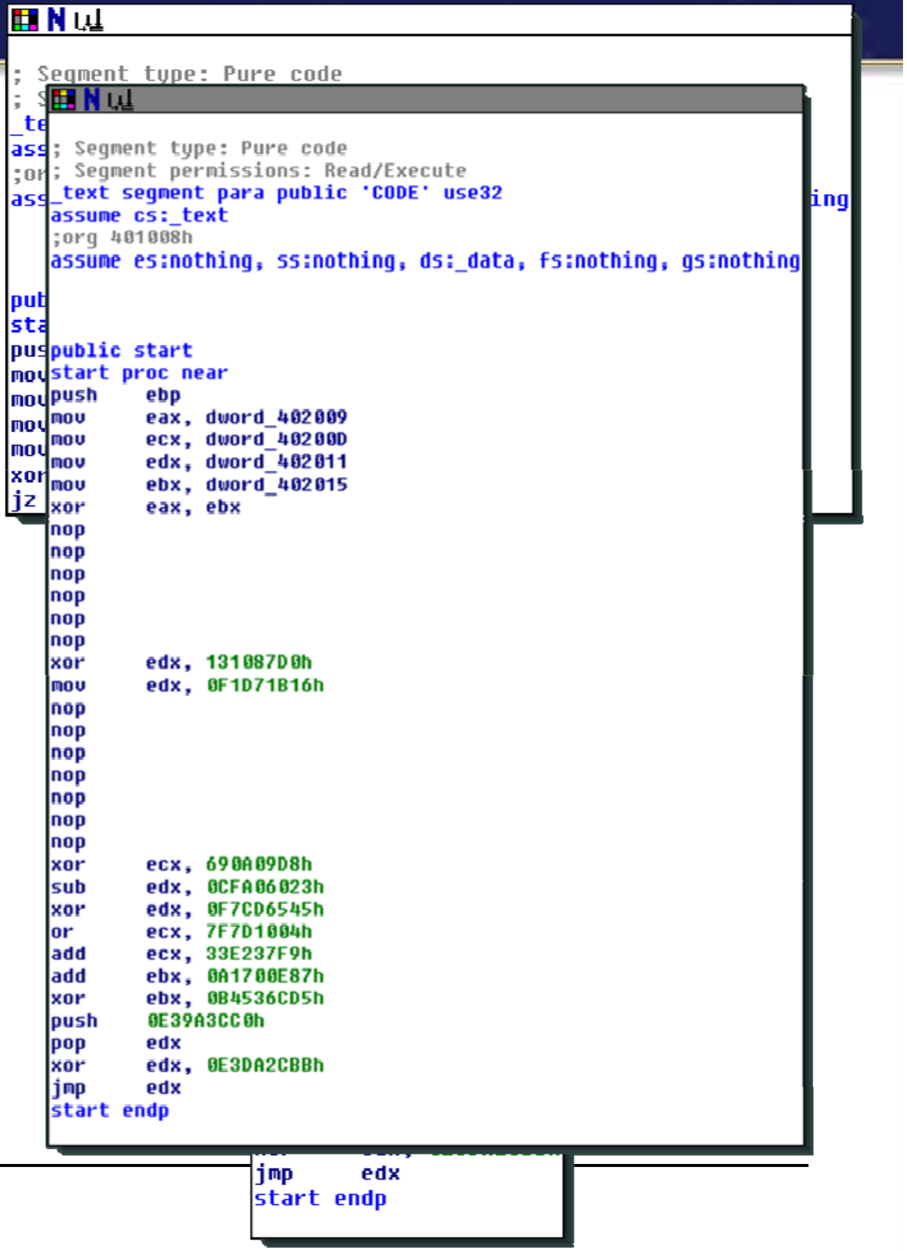
NE W! Or just modify the IDA Pro database



Patch the database?

Reload

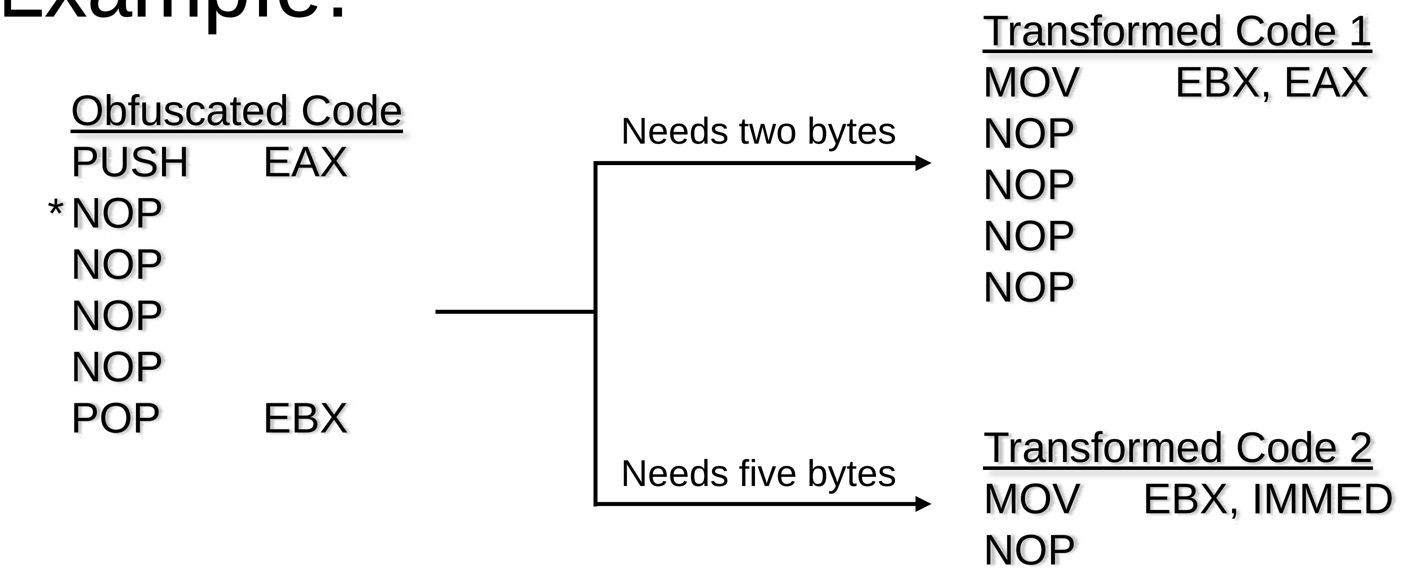
- Now, we see the obfuscated code begin to disappear
- The Deobfuscator replaces obfuscation patterns and injects NOPs over useless code to create slack space



```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use32
assume cs:_text
;org 401008h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing



public start
start proc near
push    ebp
mov     eax, dword_402009
mov     ecx, dword_40200D
mov     edx, dword_402011
mov     ebx, dword_402015
xor     eax, ebx
nop
nop
nop
nop
nop
nop
xor     edx, 131087D0h
mov     edx, 0F1D71B16h
nop
nop
nop
nop
nop
nop
nop
xor     ecx, 690A09D8h
sub     edx, 0CFA06023h
xor     edx, 0F7CD6545h
or      ecx, 7F7D1004h
add     ecx, 33E237F9h
add     ebx, 0A1700E87h
xor     ebx, 0B4536CD5h
push    0E39A3CC0h
pop     edx
xor     edx, 0E3DA2CBBh
jmp     edx
start endp
```

Slack Space

- Slack space is useful for patterns that need additional bytes to create a simplified instruction
- Example:



Obfuscated Code

```
  PUSH      EAX
* NOP
  NOP
  NOP
  NOP
  POP       EBX
```

Needs two bytes

Transformed Code 1

```
MOV       EBX, EAX
NOP
NOP
NOP
NOP
```

Needs five bytes

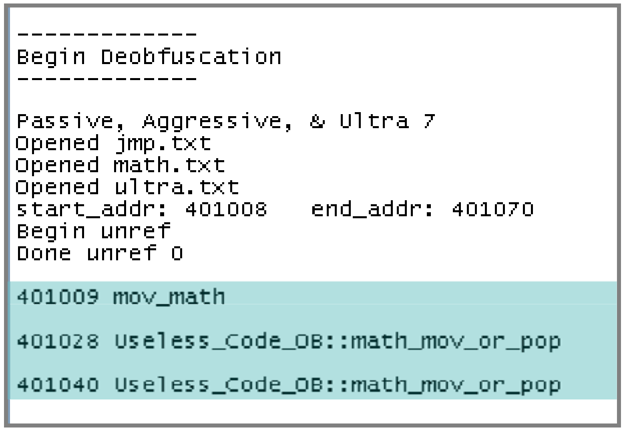
Transformed Code 2

```
MOV     EBX, IMMED
NOP
```

*Code that was removed by an earlier run of the Deobfuscator

Run 2 – Passive, Aggressive, & Ultra

- Three matching patterns
  - MOV_MATH
  - MATH_MOV_OR_POP
  - MATH_MOV_OR_POP

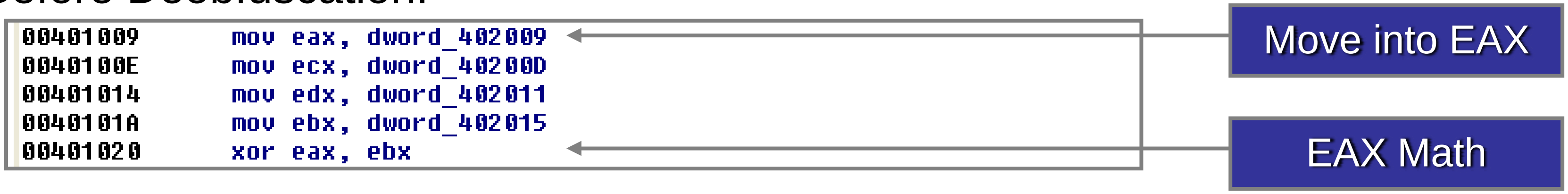


```
--------------
Begin Deobfuscation
--------------

Passive, Aggressive, & Ultra 7
Opened jmp.txt
Opened math.txt
Opened ultra.txt
start_addr: 401008    end_addr: 401070
Begin unref
Done unref 0

401009 mov_math

401028 Useless_Code_OB::math_mov_or_pop

401040 Useless_Code_OB::math_mov_or_pop
```

Pattern: *MOV_MATH*

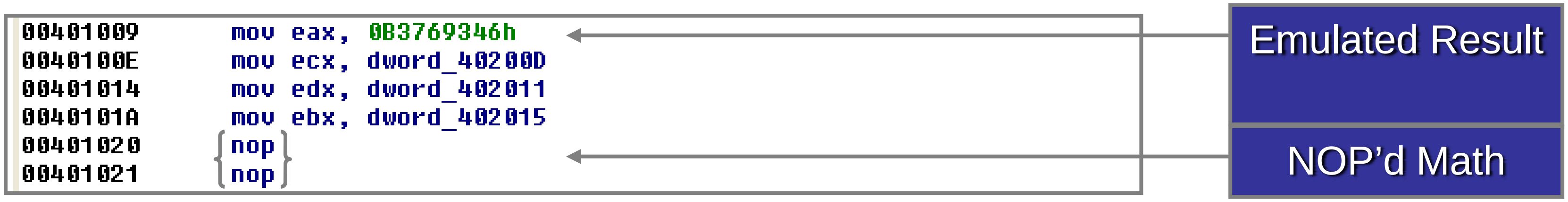
Move an immediate into EAX and XOR it with another known register value

Before Deobfuscation:



```
00401009        mov eax, dword_402009
0040100E        mov ecx, dword_40200D
00401014        mov edx, dword_402011
0040101A        mov ebx, dword_402015
00401020        xor eax, ebx
```

After Deobfuscation:

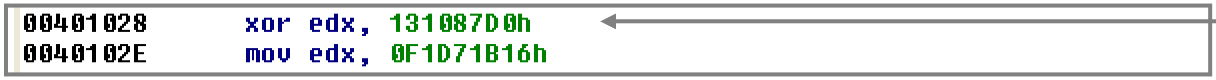


```
00401009        mov eax, 0B3769346h
0040100E        mov ecx, dword_40200D
00401014        mov edx, dword_402011
0040101A        mov ebx, dword_402015
00401020        nop
00401021        nop
```

Pattern: *MATH_MOV_OR_POP*

Do math on EDX, then MOV an immediate or POP from the stack into EDX before using it again

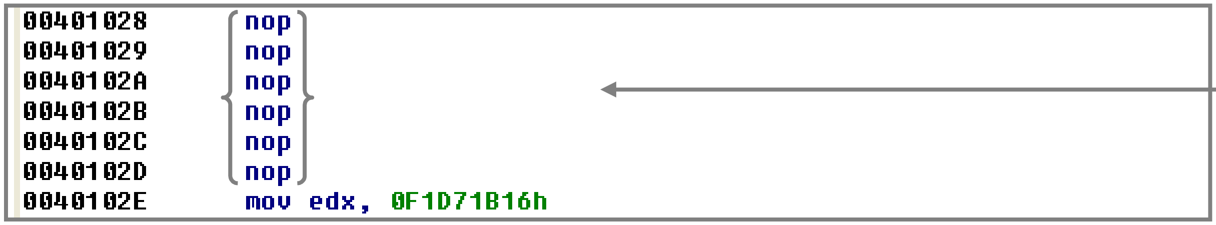
Before Deobfuscation:



```
00401028        xor edx, 131087D0h
0040102E        mov edx, 0F1D71B16h
```

EDX Math

After Deobfuscation:



```
00401028        nop
00401029        nop
0040102A        nop
0040102B        nop
0040102C        nop
0040102D        nop
0040102E        mov edx, 0F1D71B16h
```

NOP’d Math

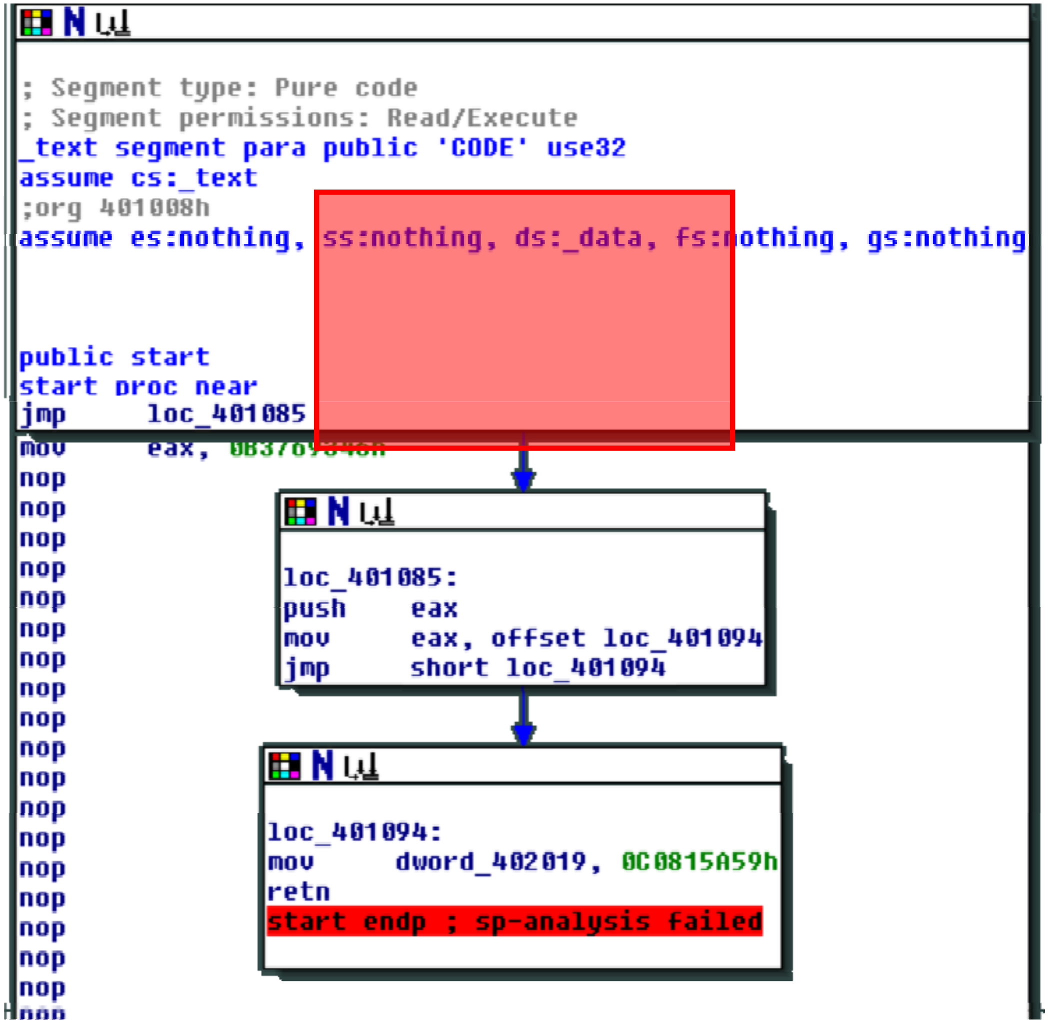
Finishing Up

- The Deobfuscator has finished matching obfuscation patterns
- Slack space is no longer needed, so we run one of the clean-up modes to simplify the appearance of the control flow
- “NOP Remove” **injects JMPs** to remove NOPs from control flow
- “Collapse” mode moves code to slack space to **eliminate NOPs and JMPs**

NOP Remove

Before:

After:



```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use32
assume cs:_text
;org 401008h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing



public start
start proc near
jmp     loc_401085
mov     eax, 0B3769848h
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```

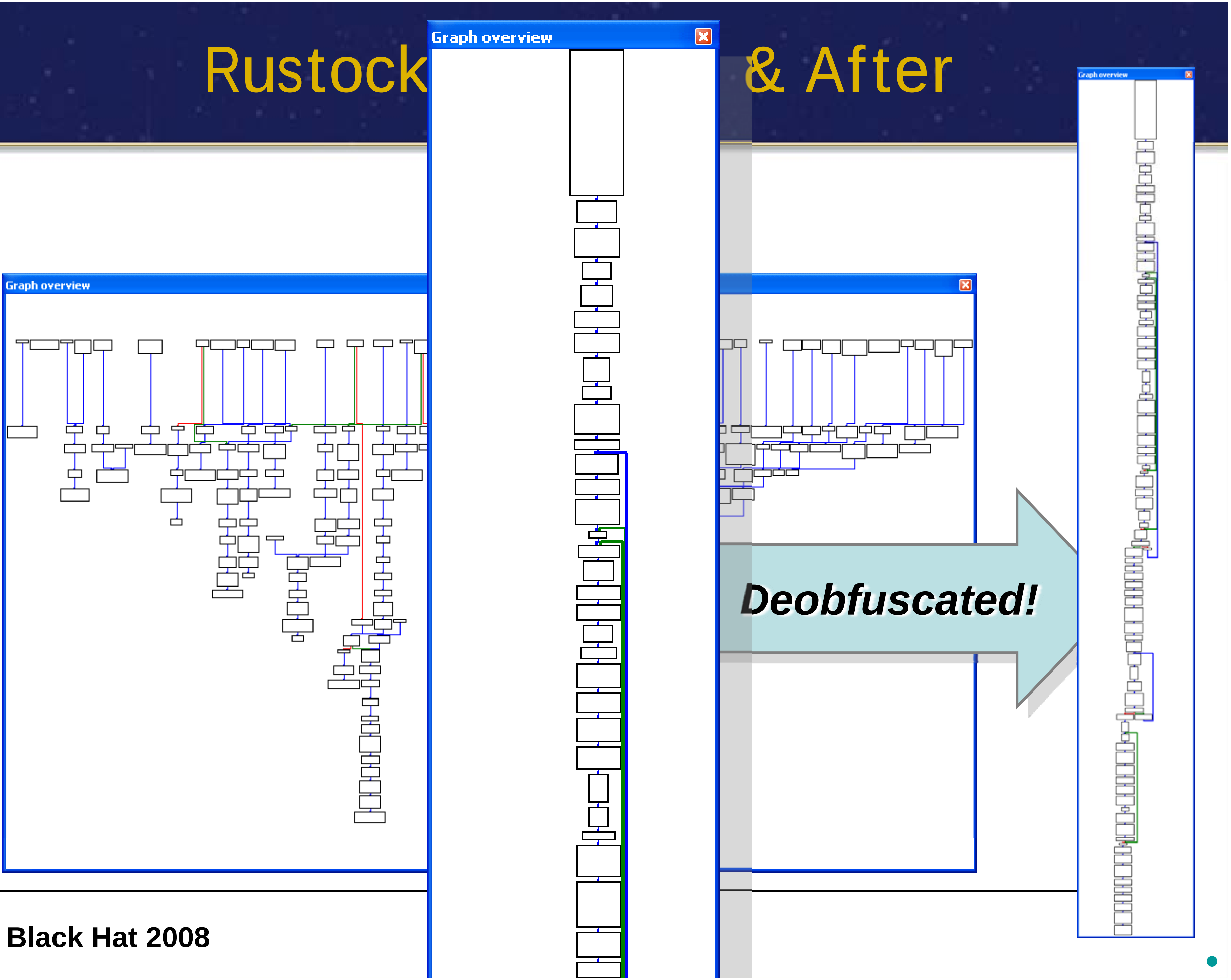
```
loc_401085:
push    eax
mov     eax, offset loc_401094
jmp     short loc_401094
```

```
loc_401094:
mov     dword_402019, 0C0815A59h
retn
start endp ; sp-analysis failed
```

Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

Rustock & After



RustockB Decryption Pseudo-code

```
for (i = 7; i > 0; i--)
{
    Address = 0x00401B82      // Starting address of encrypted region
    Key1 = 0x4DFEE1C0         // Decryption key 1
    Key2 = 0x0869ECC5         // Decryption key 2
    Key3 = 0                  // Decryption key 3
    Key4 = 0                  // Decryption key 4 (Accumulator)
    for (j = 0x44DC; j > 0; j--, Address += 4)    // 0x44DC = size of encrypted region
    {
          for (k = 2; k > 0; k--)
          {
                    Key4 = k * 4
                    XOR Key4, 0x5E57B7DE
                    XOR Key4, Key3
                    Key4 += Key2
                    XOR Key1, k
                    [Address] -= Key4
                    Key3 += Key1
          }
    }
}

for (i = 0x44DC, Address = 0x00401B82, Sum = 0; i > 0; i--, Address += 4)
    Sum += [Address]          // Add up the encrypted region (a DWORD at a time) in EAX

for (i = 0x44DC, Address = 0x00401B82; i > 0; i--, Address += 4)
    XOR [Address], Sum        // XOR each DWORD of the encrypted region with the sum in EAX
```

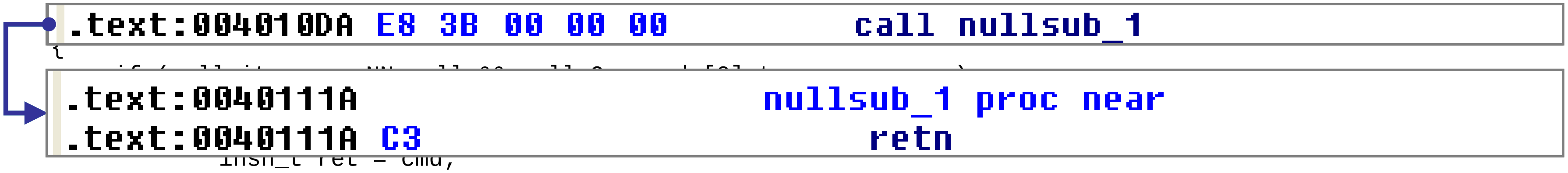
Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

Sample Source Code

The Simple Solution:

A Simple Problem:



```
.text:004010DA E8 3B 00 00 00          call nullsub_1
.text:0040111A                    nullsub_1 proc near
.text:0040111A C3                       retn
```

```
//---------------------------------------------------------------------------------
// CALL NULL - A function call that just returns
//---------------------------------------------------------------------------------
{
    if (call.itype == NN_call && call.Operands[0].type == o_near)
            insn_t ret = cmd;

            // Function that just returns
            if (ret.itype == NN_retn)
            {
                        *instr_offset = call.size;
                        msg("\n%a CALL_NULL\n", call.ea);

                        // NOP the call
                        fprintf(outfile, "%X 5 90 90 90 90 90\n", get_fileregion_offset(call.ea));

                        // NOP the return
                        fprintf(outfile, "%X 1 90\n", get_fileregion_offset(ret.ea));

                        return 1;
            }
    }

    return 0;
}
```

Overview

- The Problem: Obfuscation
- Malware Example: RustockB
- The Solution: Deobfuscator
- Demonstration
- RustockB: Before & After
- Sample Source Code
- Summary

Summary

- Most malware authors that wish to protect their IP use obfuscation techniques
- The Deobfuscator detects and simplifies many of these obfuscation and anti-disassembly patterns
- Over time, the repository of patterns will be developed to characterize most generic cases of obfuscation

Future Development

- Iterative patching of IDA database

Future Development

- ✓Iterative patching of IDA database
- Code collapsing

Future Development

- ✓ Iterative patching of IDA database
- ✓ Code collapsing
- Grammar
- Black-box control flow

Contact

- For more information on this and other tools, contact:

Eric Laspe, Reverse Engineer
elaspe@rri-usa.org
937-427-7042

Jason Raber, Lead Reverse Engineer
jraber@rri-usa.org
937-427-7085

- Visit us online:
  http://www.rri-usa.org/isrsoftware.html