Patchwork

Stitching against malware families with IDA Pro

Daniel Plohmann

plohmann@cs.uni-bonn.de

Fraunhofer FKIE

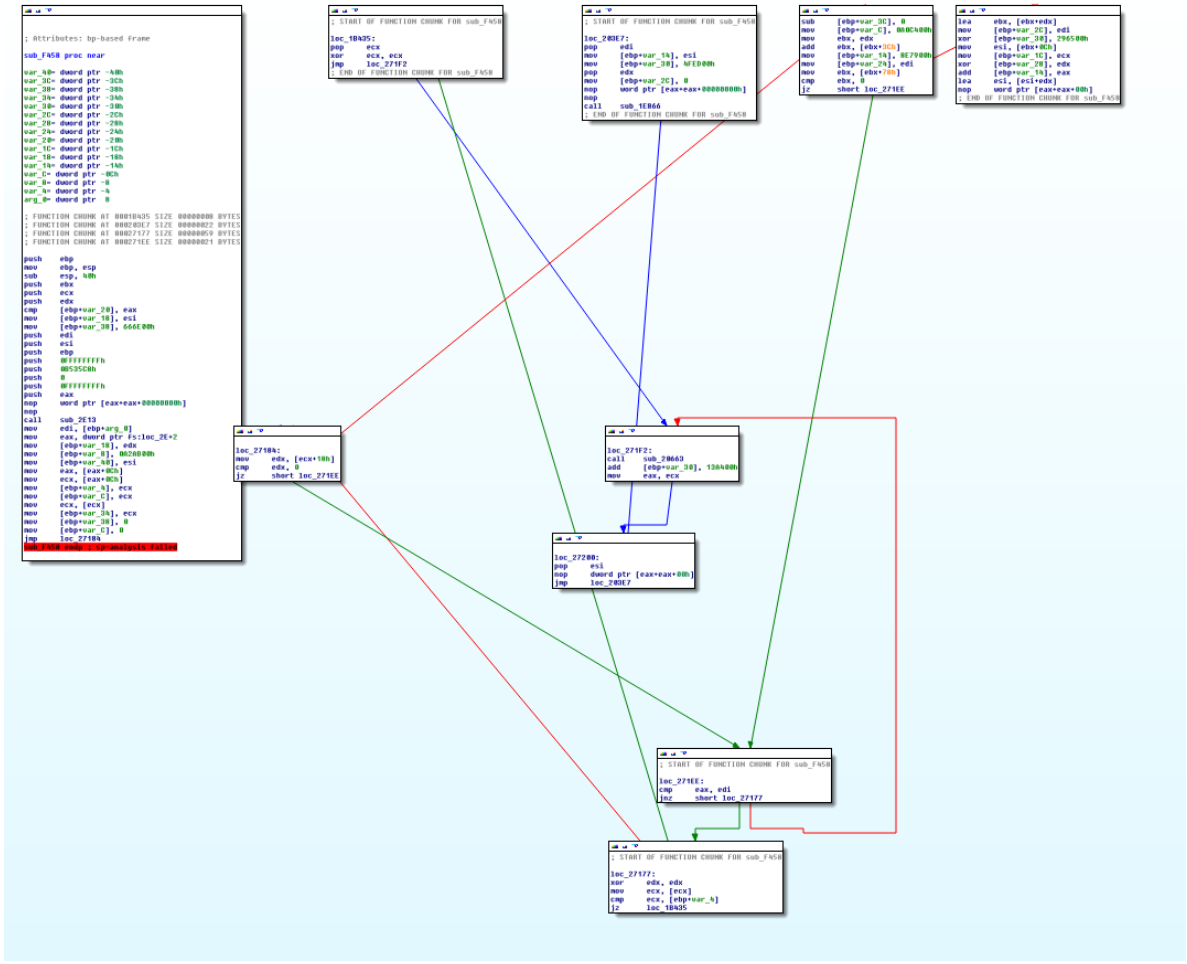
Some words about myself

- Personal background
  - PhD student and researcher at University of Bonn & Fraunhofer FKIE
    - Research focus: Efficiency of Reverse Engineering
    - Work focus: malware analysis and botnet mitigation

- Related projects
  - [1] PyBox (python sandboxing toolkit)
  - [2] IDAscope (IDA Pro enhancements for malware RE)

[1] http://code.google.com/p/pyboxed
[2] https://idascope.pnx.tf

Patchwork
Motivation



Patchwork
… in a nutshell

- Patchwork = refurbished (IDA)PyEmu [1] + set of convenience functions


Work in progress

- Driving ideas:
  - A flexible framework that allows data transformations aiding static analysis
  - Get away from the (throwaway-)snippets-per-case approach


Sharing is caring!

[1] https://github.com/codypierce/pyemu (2009 / 2012)

Patchwork
Wishlist

- Seamless integration with IDA
- Instrumentalization of analysis target's native code
  - But don't actually run code (= no debugging, AppCall, PIN, …)
- Reusability / generalization

- Notable emulation solutions compatible with IDA:
  - [1] Ida-x86emu: standalone plugin, no extendability
  - [2] (IDA)PyEmu: python-based, fully scriptable
    - Incomplete (limited to most common opcodes)

      Outdated (state of 2009)

[1] http://www.idabook.com/ida-x86emu
[2] https://github.com/codypierce/pyemu

(IDA)PyEmu
Workflow

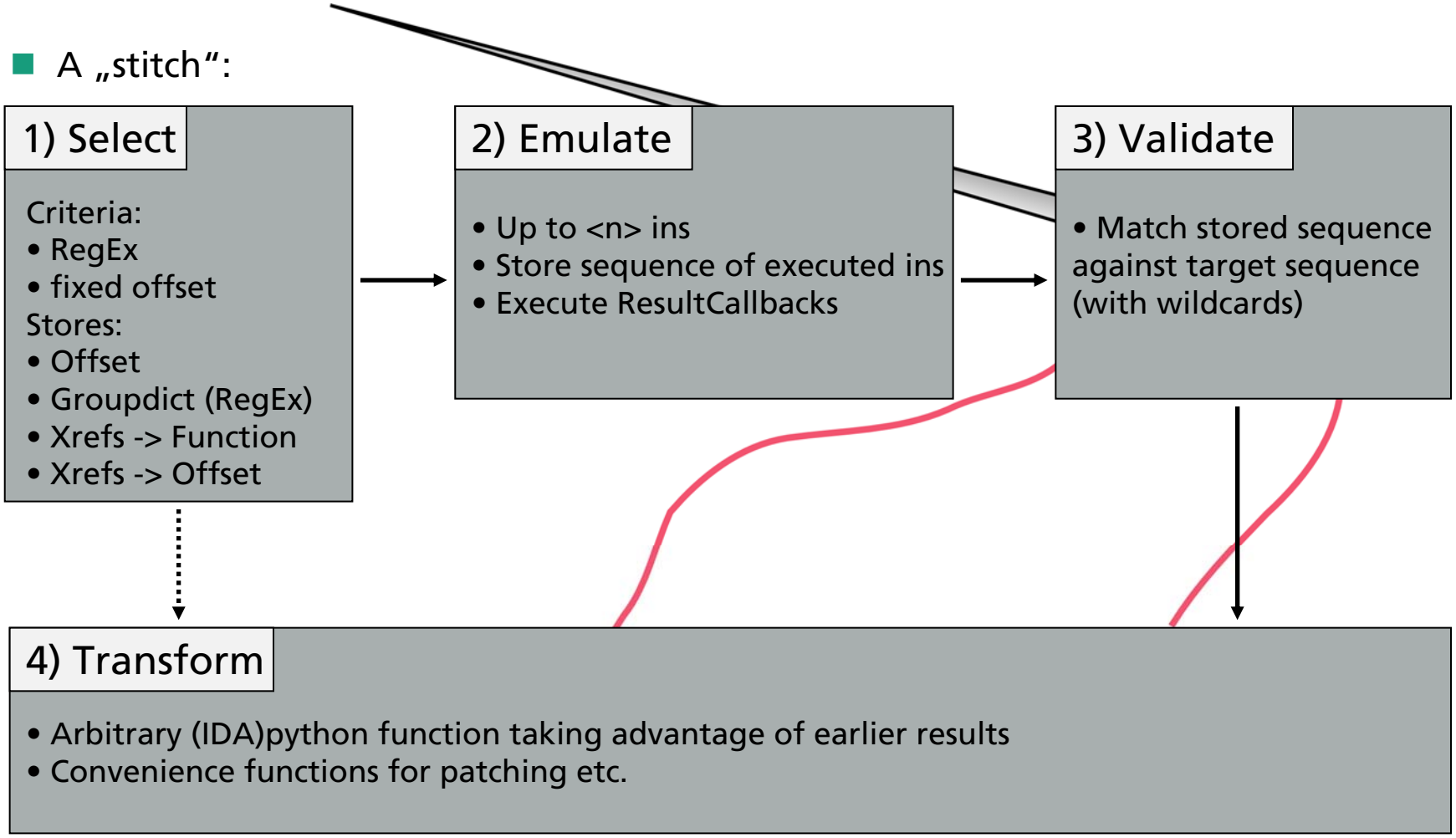
- Pretty straight-forward :)

- Set initial emulation state
  - Allocate + fill virtual memory
  - Create context (stack / registers + EIP)
- Emulate step by step

  Proceed with modified state (algorithmic results, transformed memory)

Patchwork
Workflow

- A „stitch“:



Patchwork
Example: Nymaim

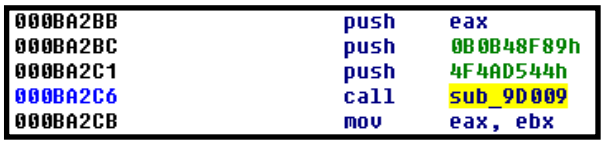
- Dropper / Ransom malware family [1]
- Written in assembler, heavily obfuscated
  - Control flow obfuscation (call/jmp redirection)
  - Obfuscated stack/register usage (delegated to subfunction)
  - Obfuscated stack usage (introduction of many irrelevant fields)

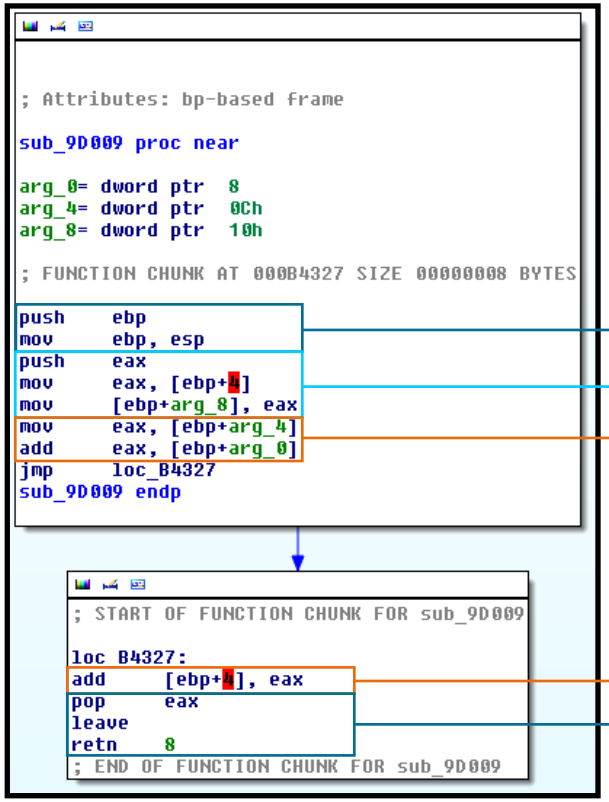
  Hashed API calls

[1] http://www.welivesecurity.com/2013/08/26/nymaim-obfuscation-chronicles/

Patchwork

Example: Nymaim (Control Flow Obfuscation)





Arg_0: Displacement offset part 0 (0x4F4AD544)

Arg_4: Displacement offset part 1 (0xB0B48F89)

Arg_8: Placeholder for original return address

Function prologue

Save original return address

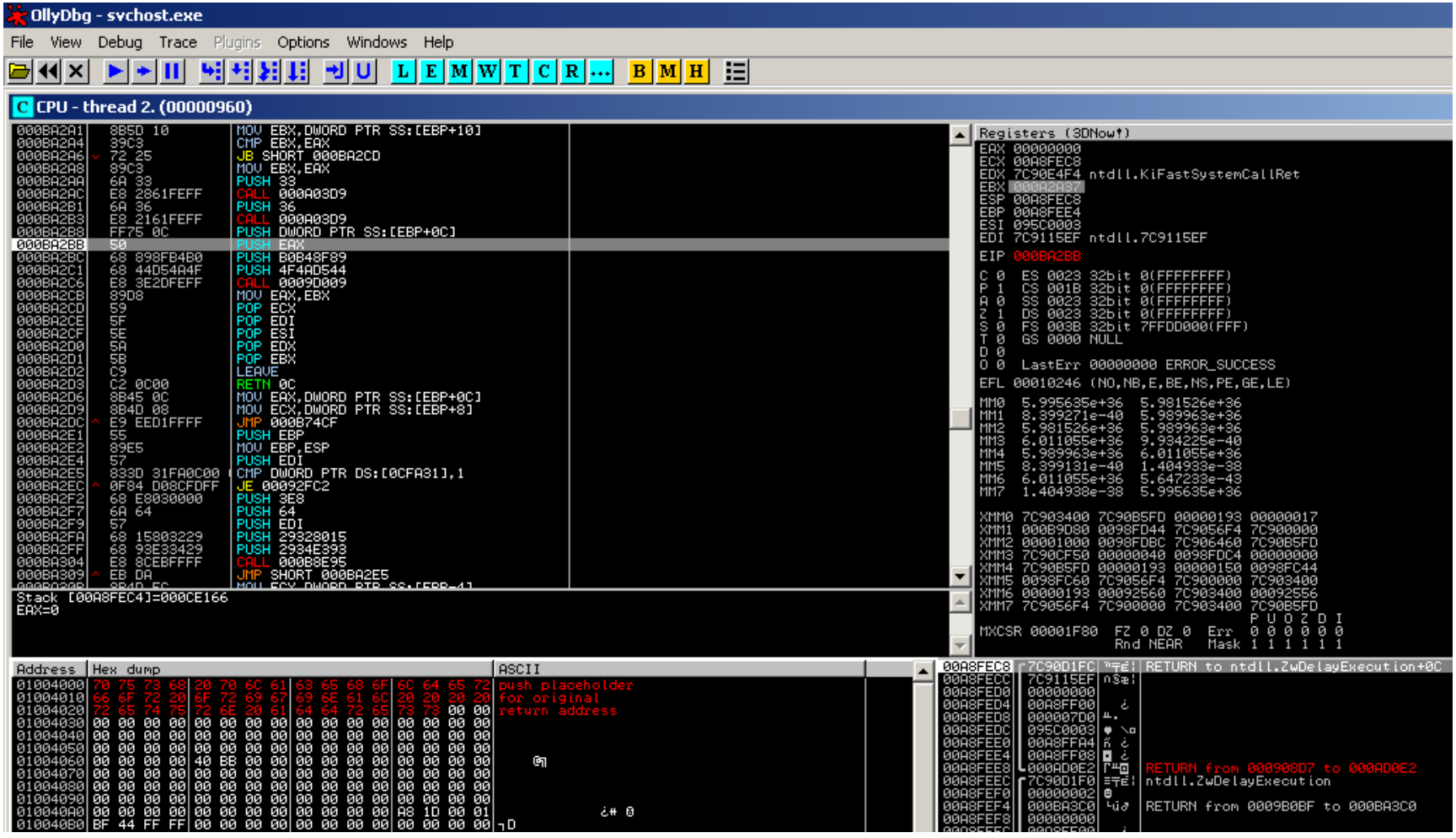
Calculate displacement

Apply displacement to return address

Clean up and detour to deplaced return address

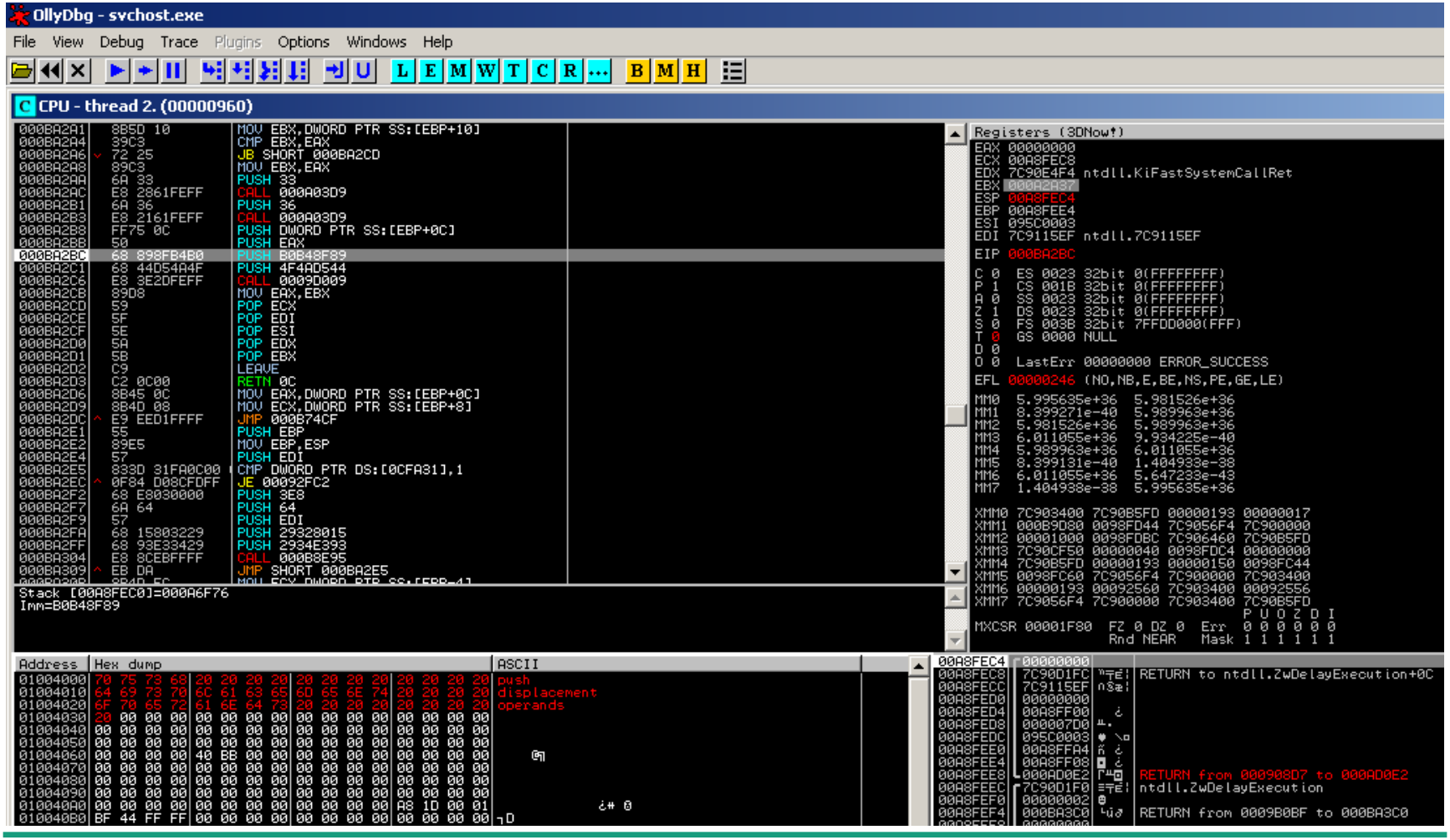
Patchwork

Example: Nymaim (Control Flow Obfuscation)



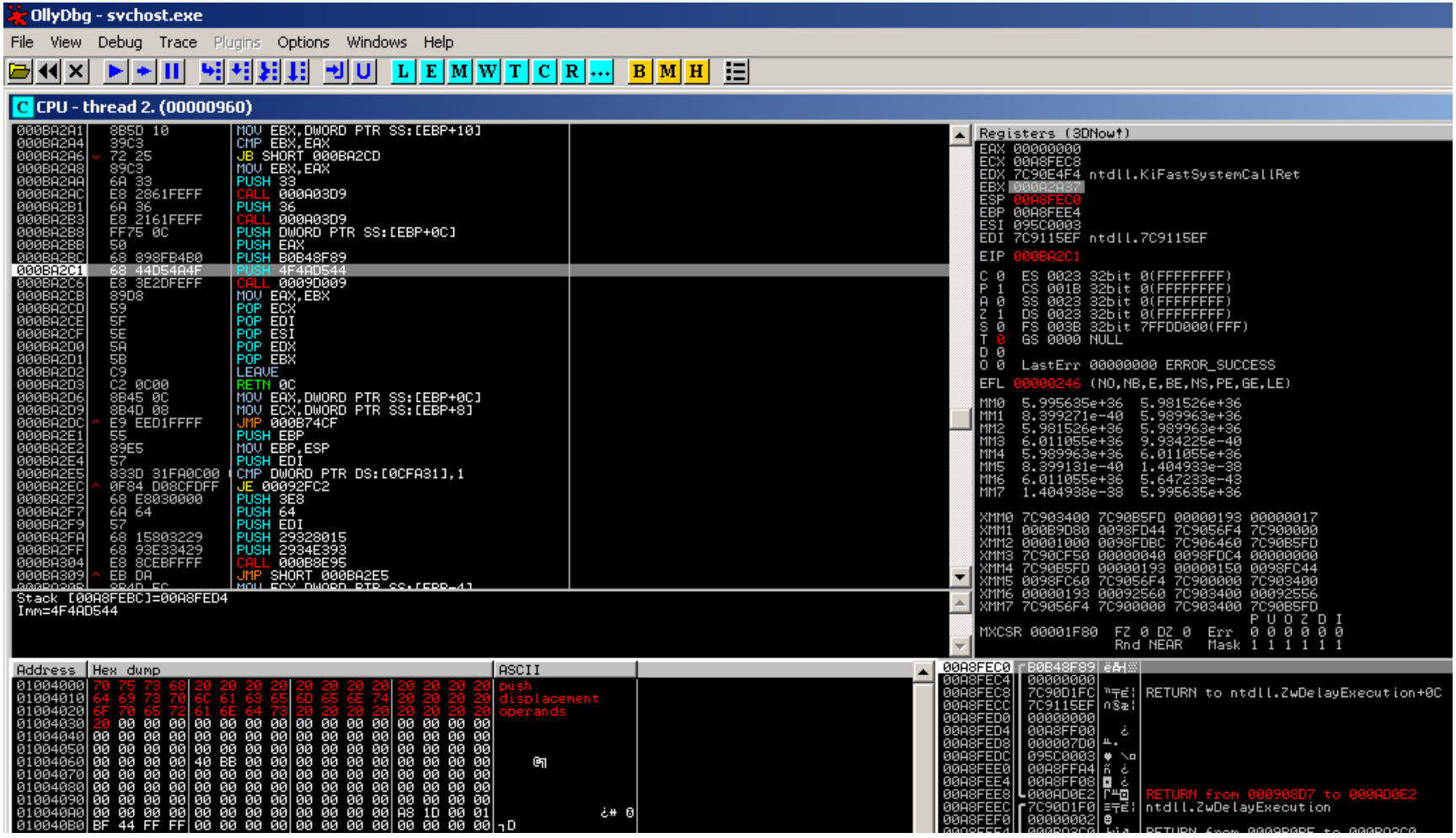
Patchwork

Example: Nymaim (Control Flow Obfuscation)



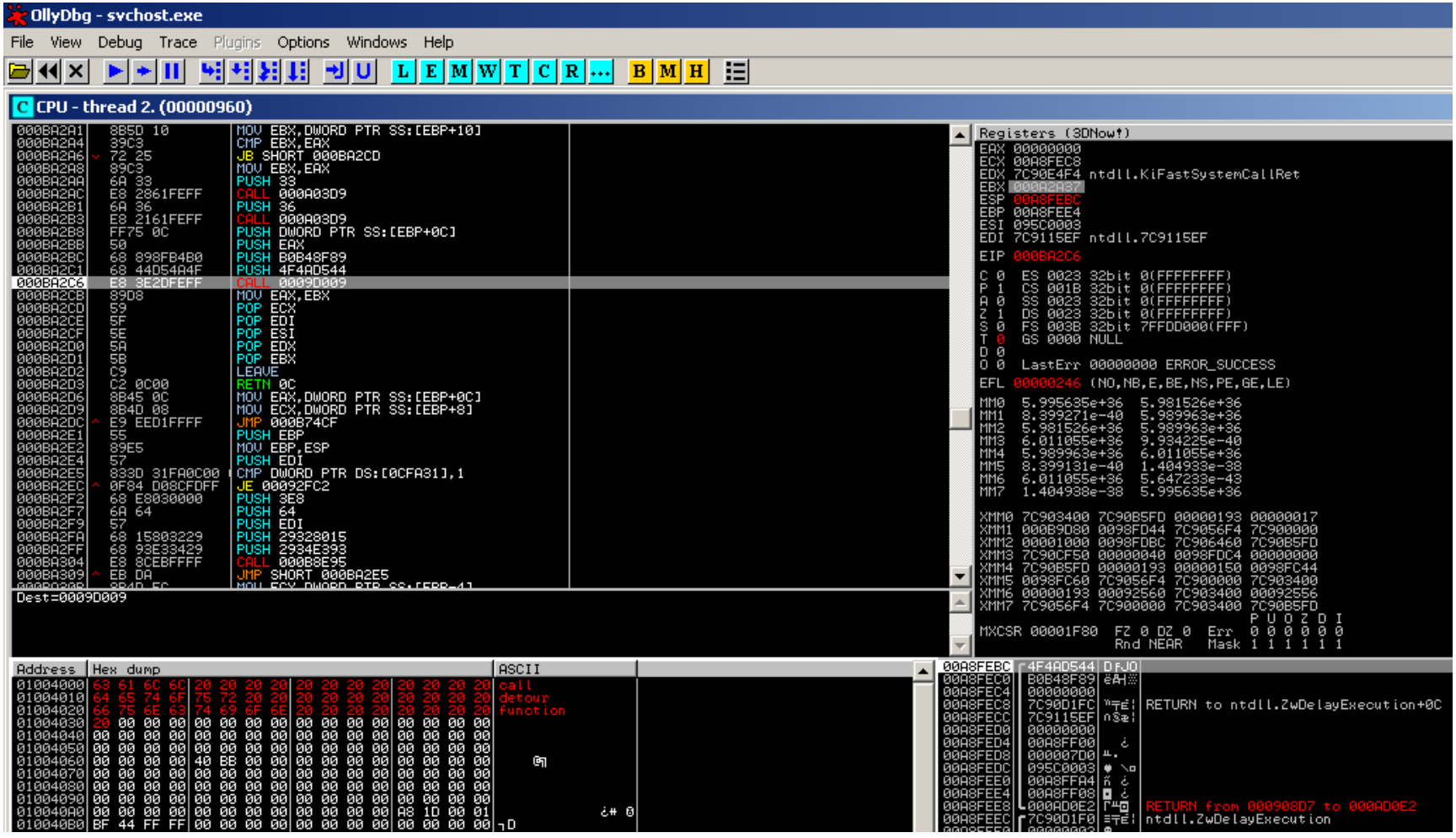
Patchwork

Example: Nymaim (Control Flow Obfuscation)



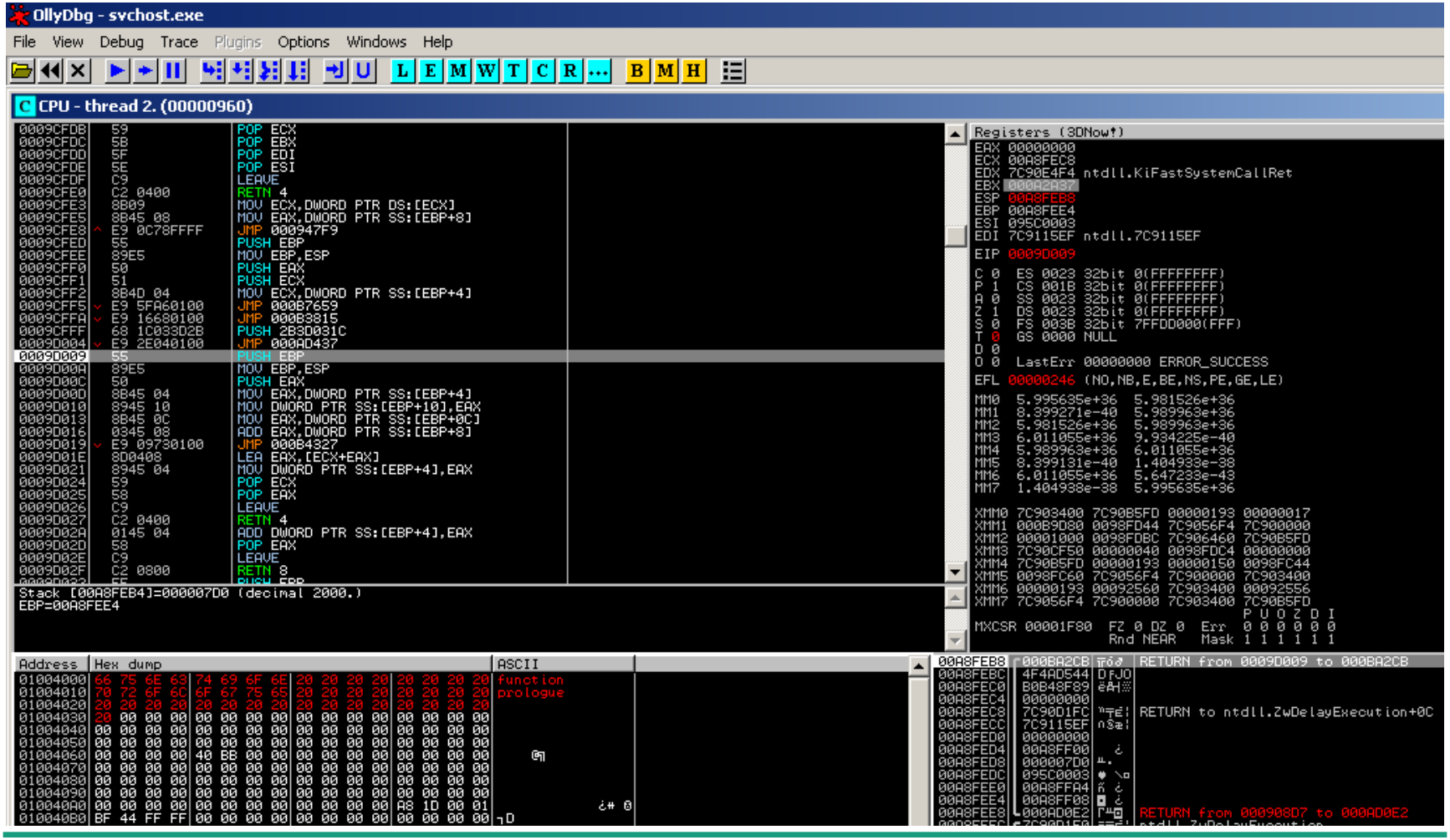
Patchwork

Example: Nymaim (Control Flow Obfuscation)



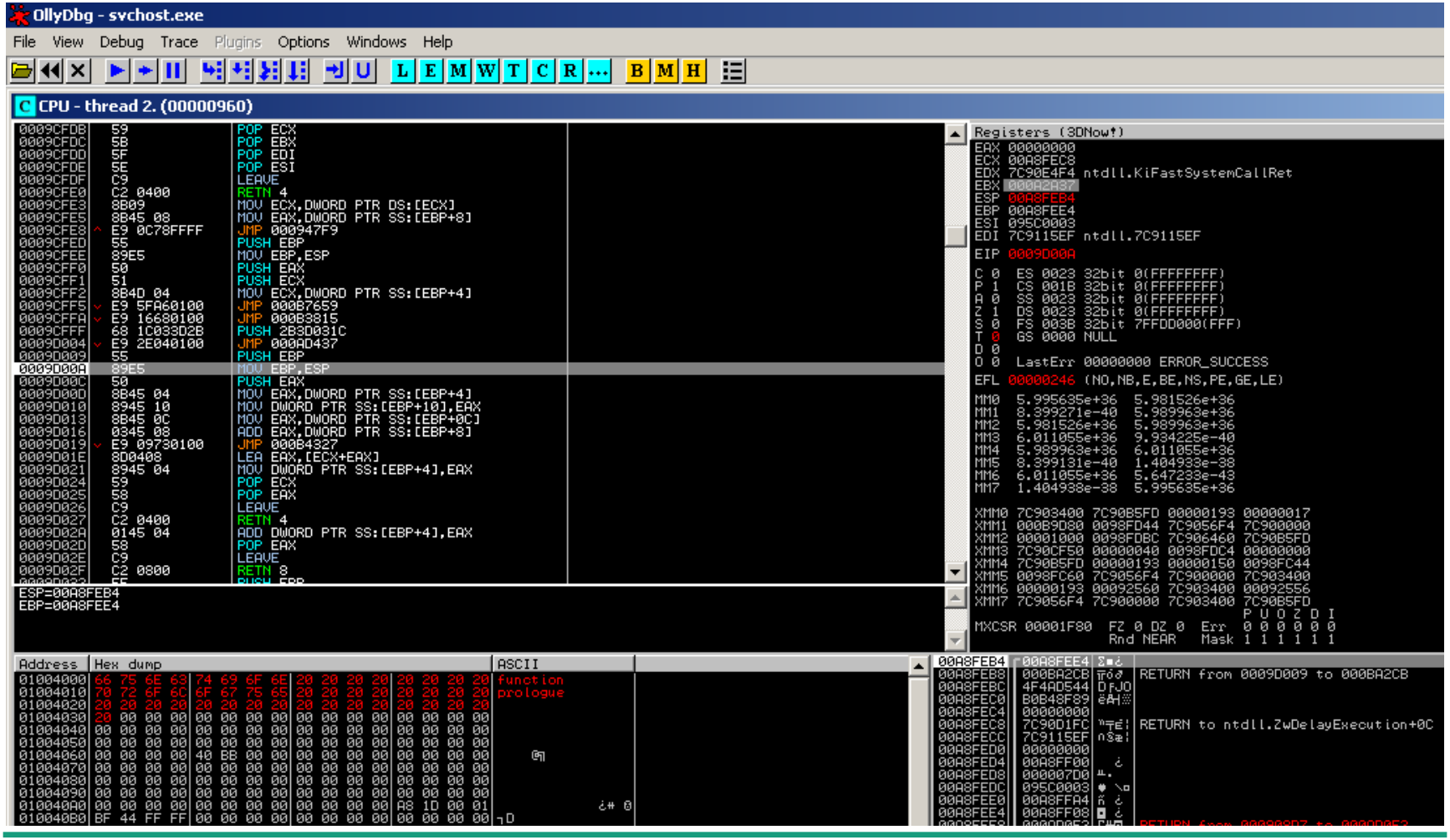
Patchwork

Example: Nymaim (Control Flow Obfuscation)



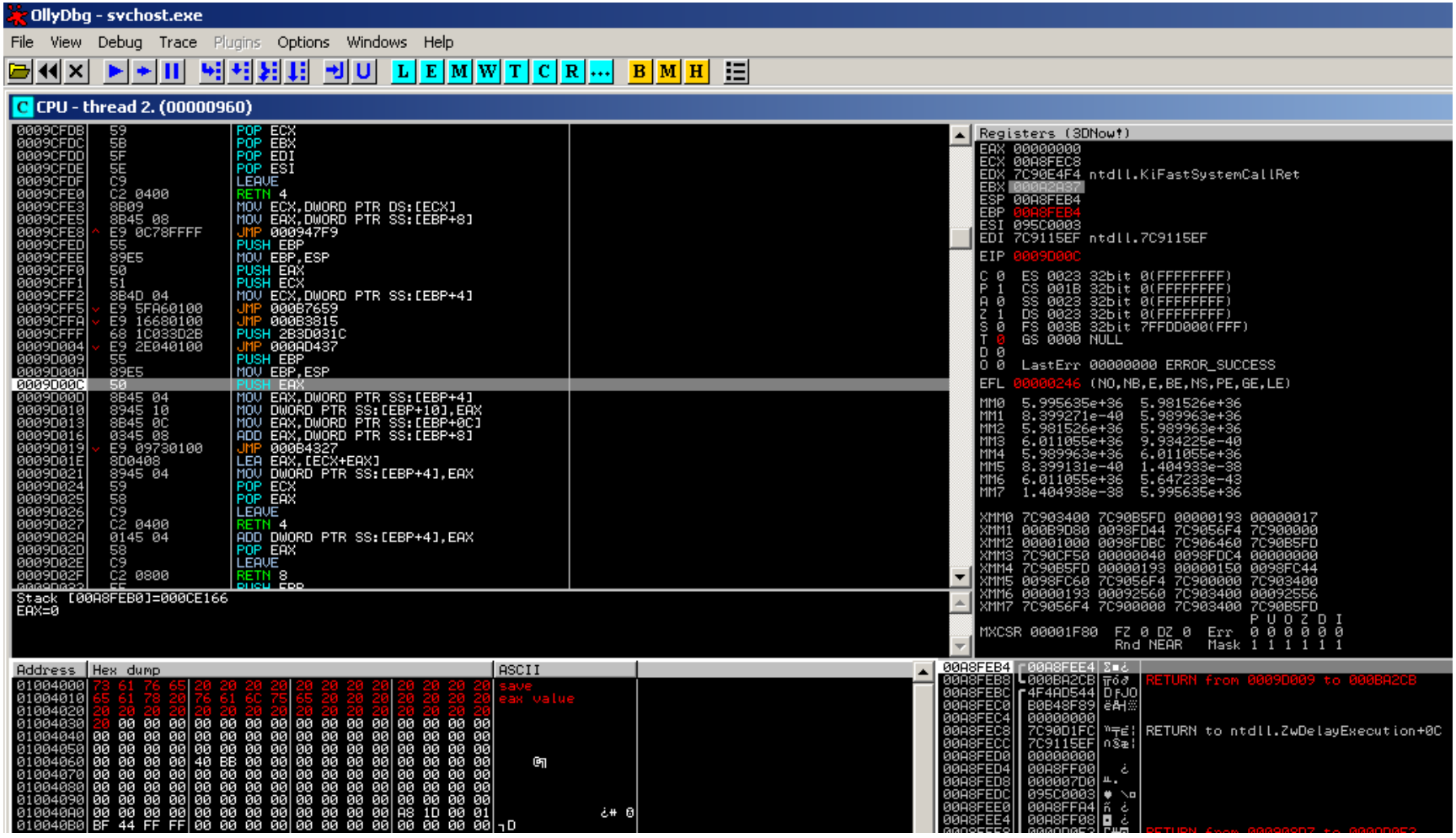
Patchwork

Example: Nymaim (Control Flow Obfuscation)



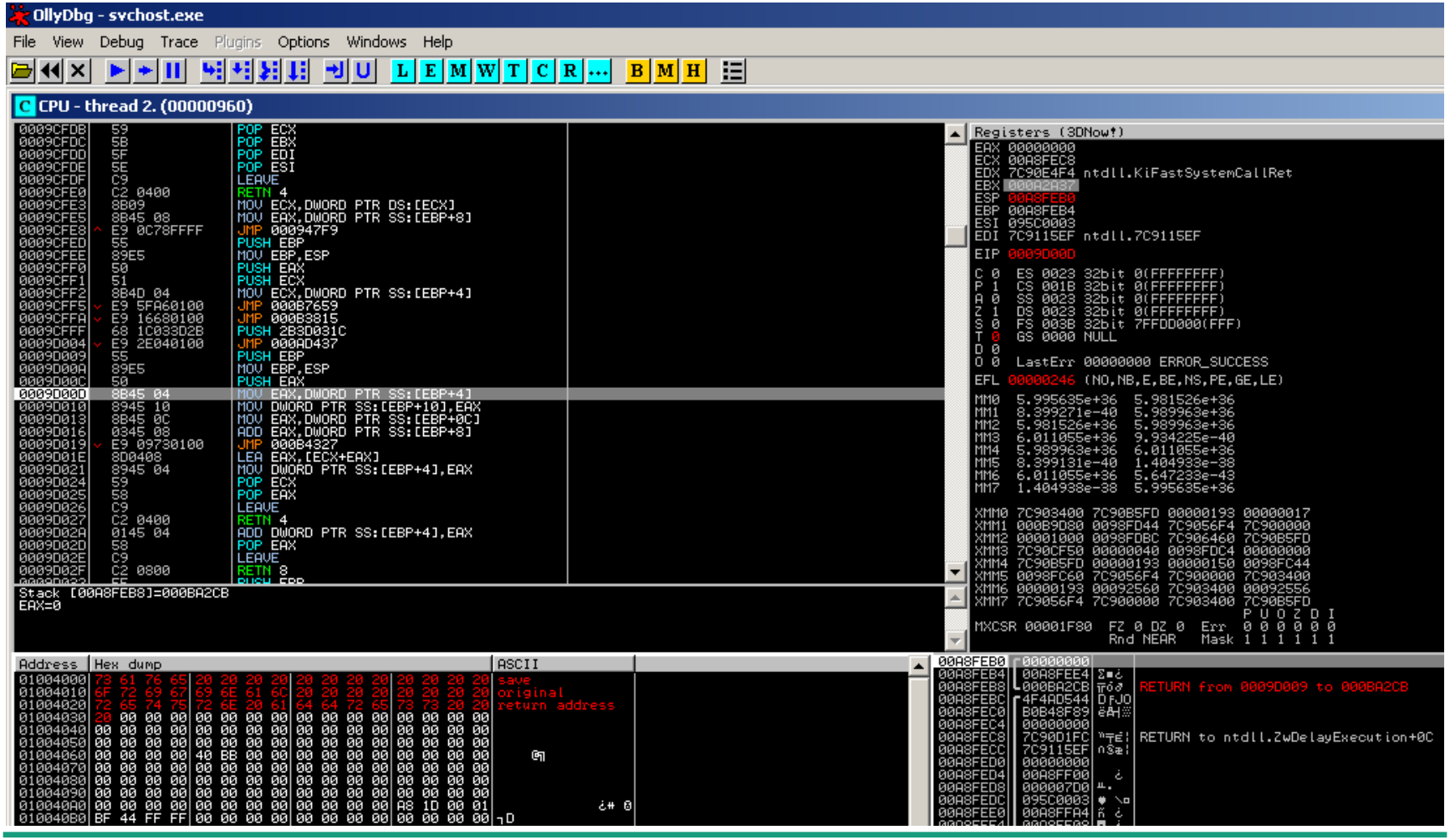
Patchwork

Example: Nymaim (Control Flow Obfuscation)



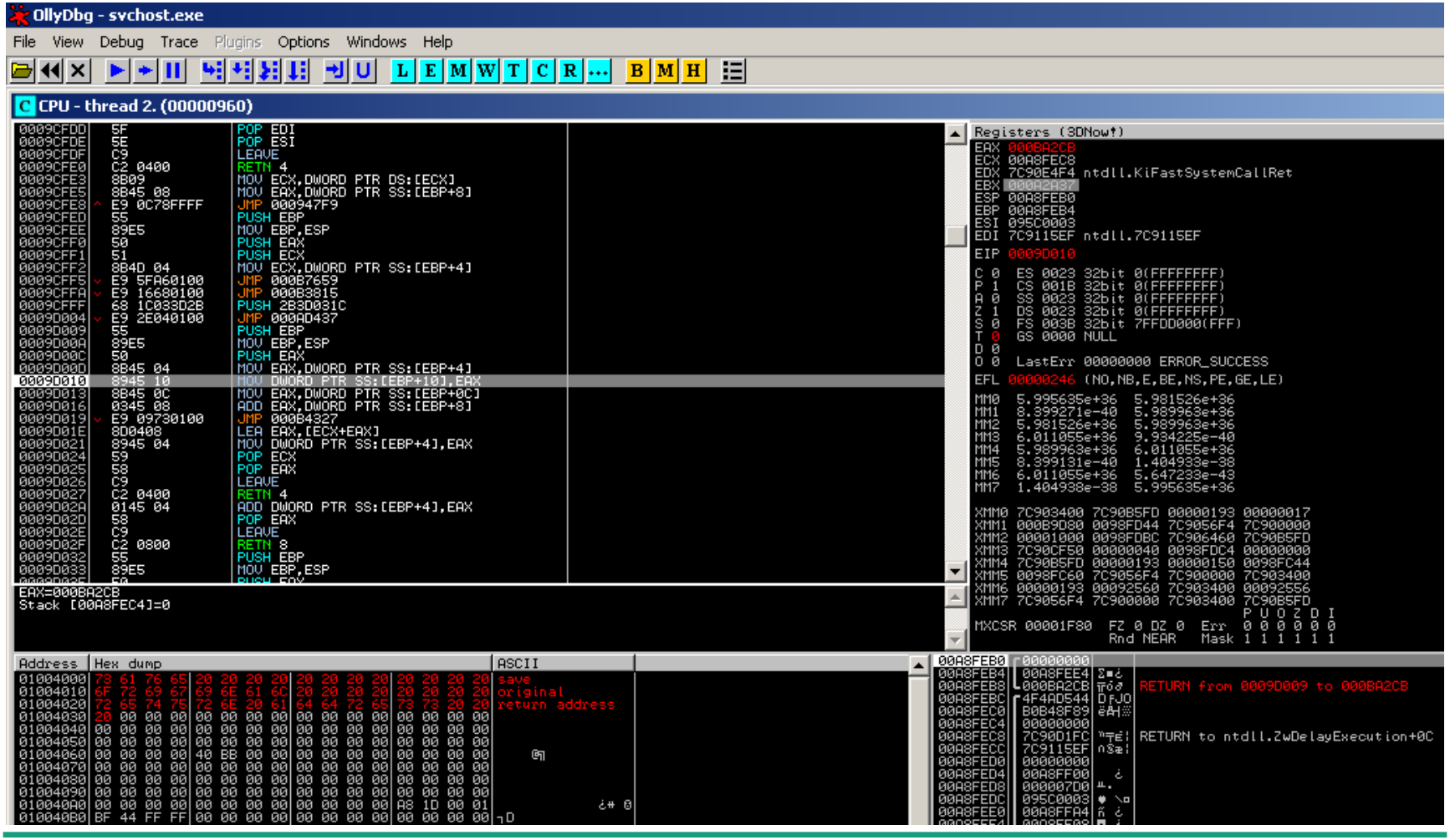
Patchwork

Example: Nymaim (Control Flow Obfuscation)



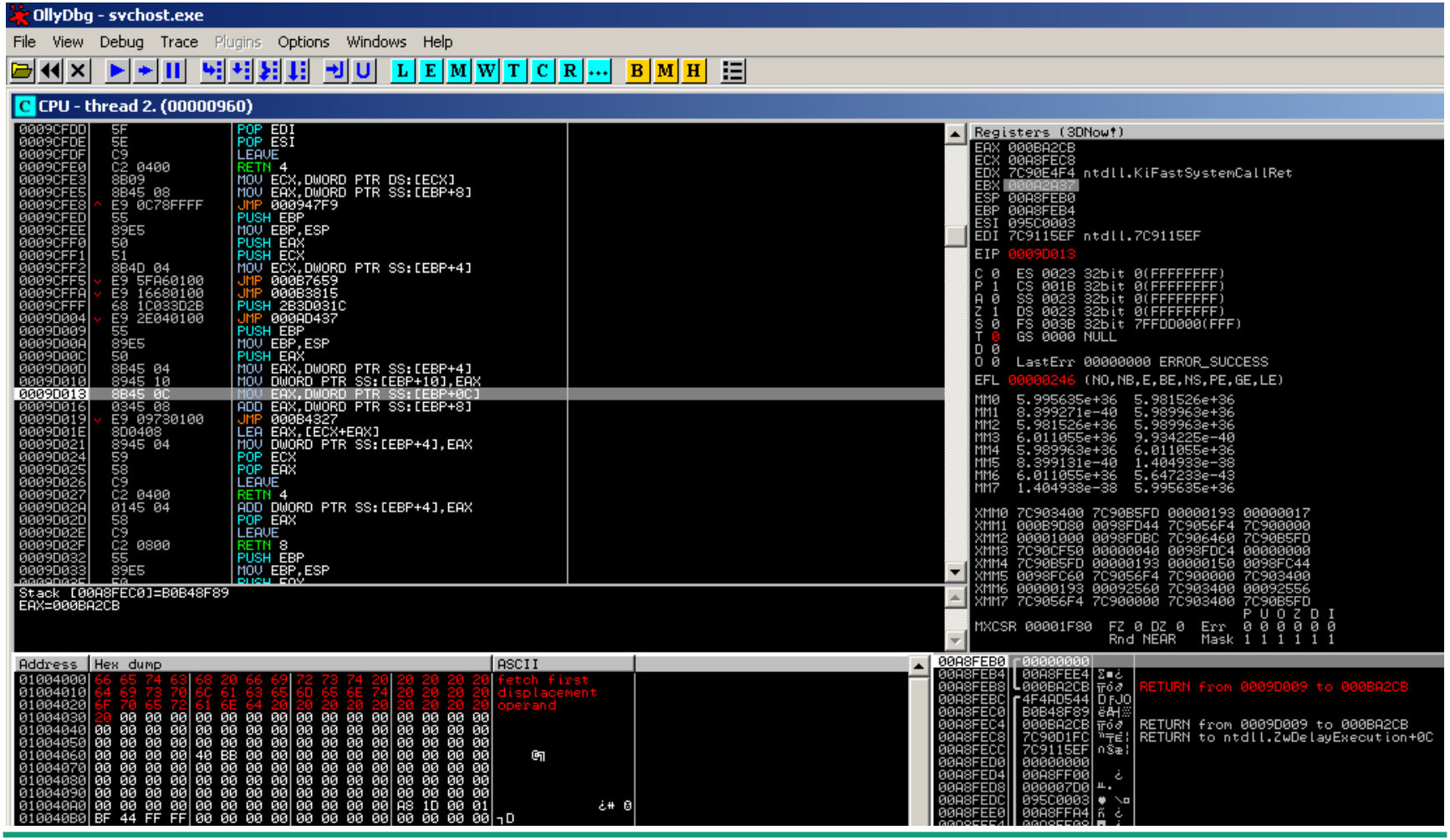
Patchwork

Example: Nymaim (Control Flow Obfuscation)



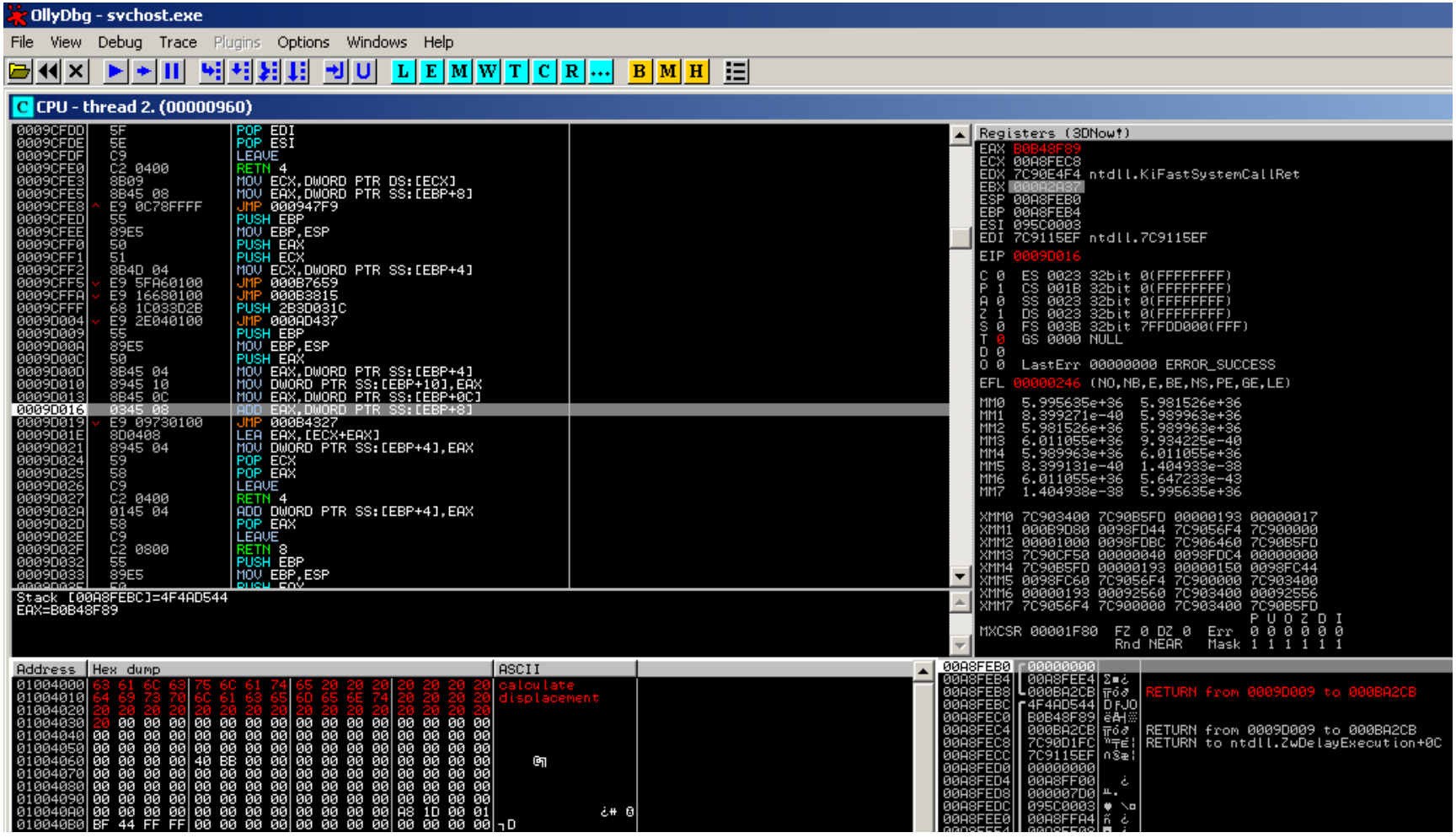
Patchwork

Example: Nymaim (Control Flow Obfuscation)



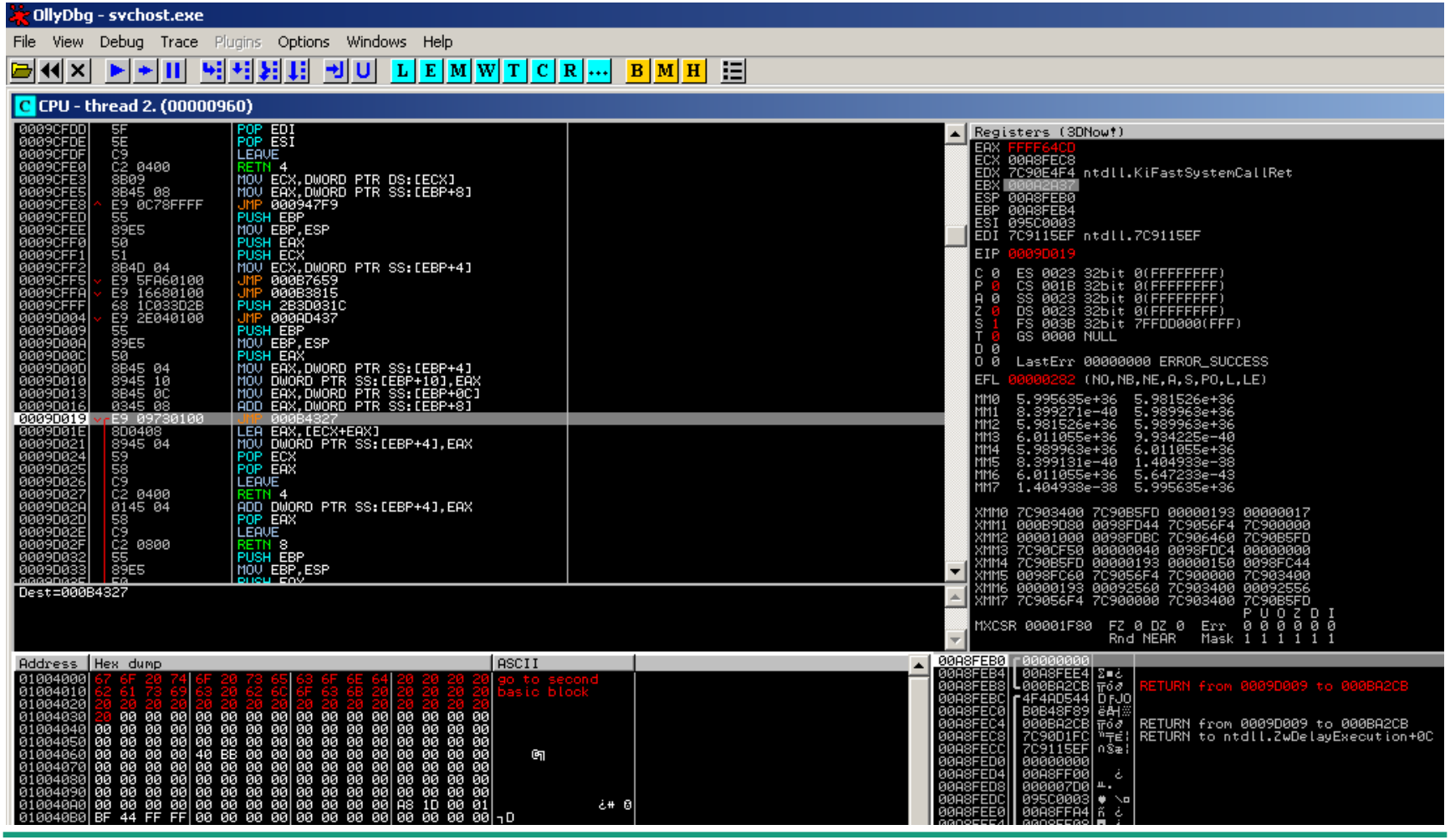
Patchwork

Example: Nymaim (Control Flow Obfuscation)



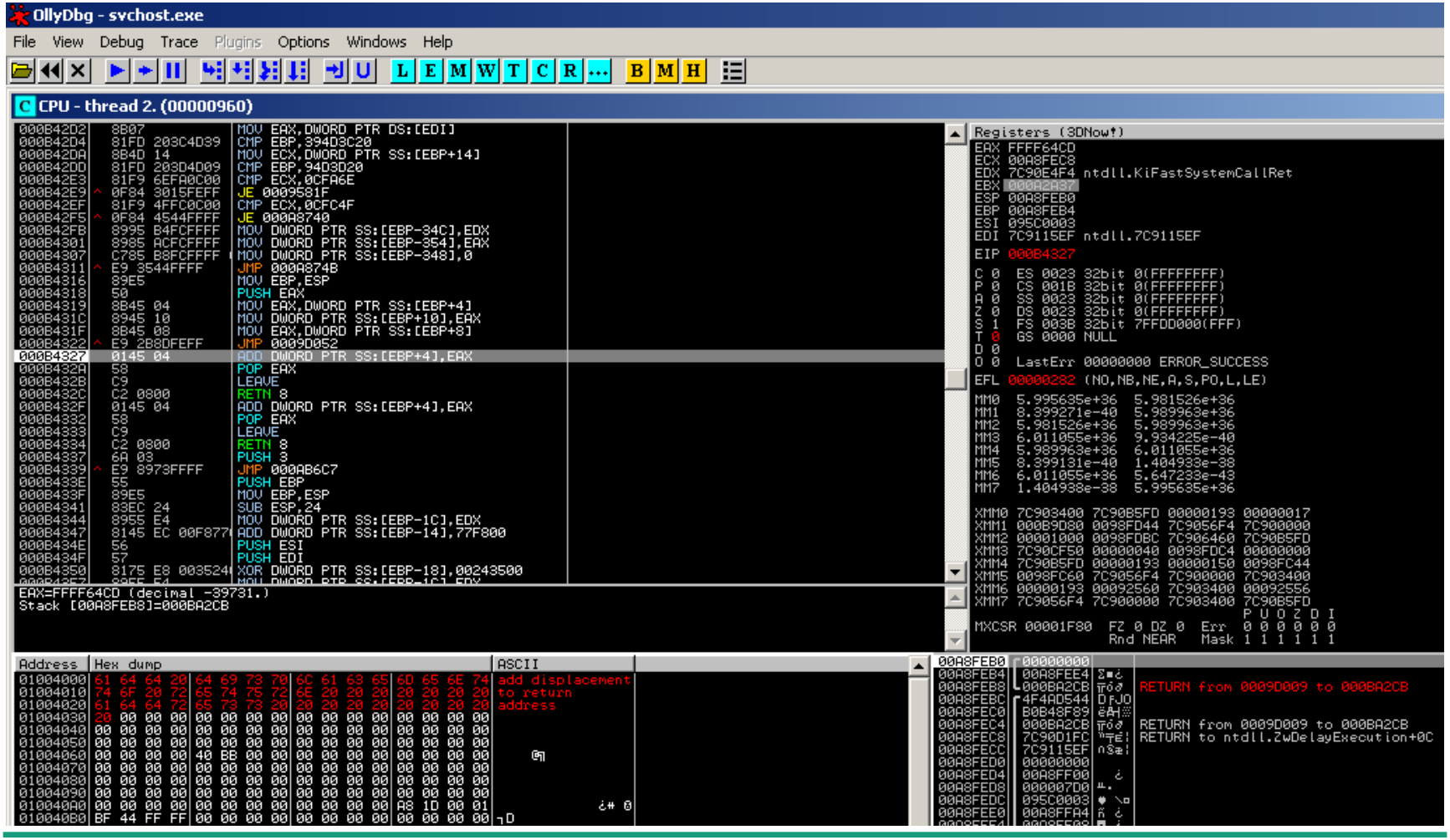
Patchwork

Example: Nymaim (Control Flow Obfuscation)



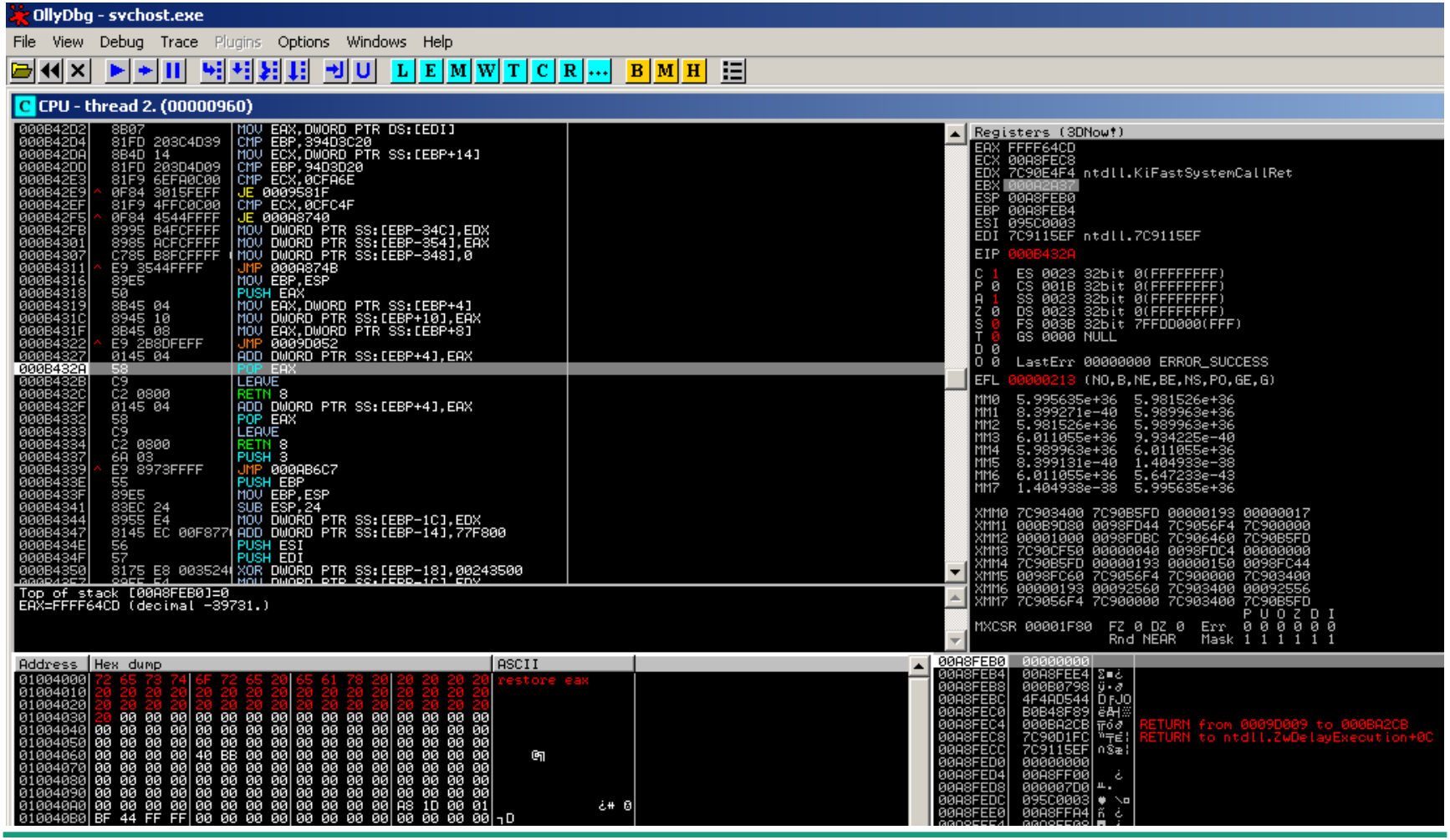
Patchwork

Example: Nymaim (Control Flow Obfuscation)



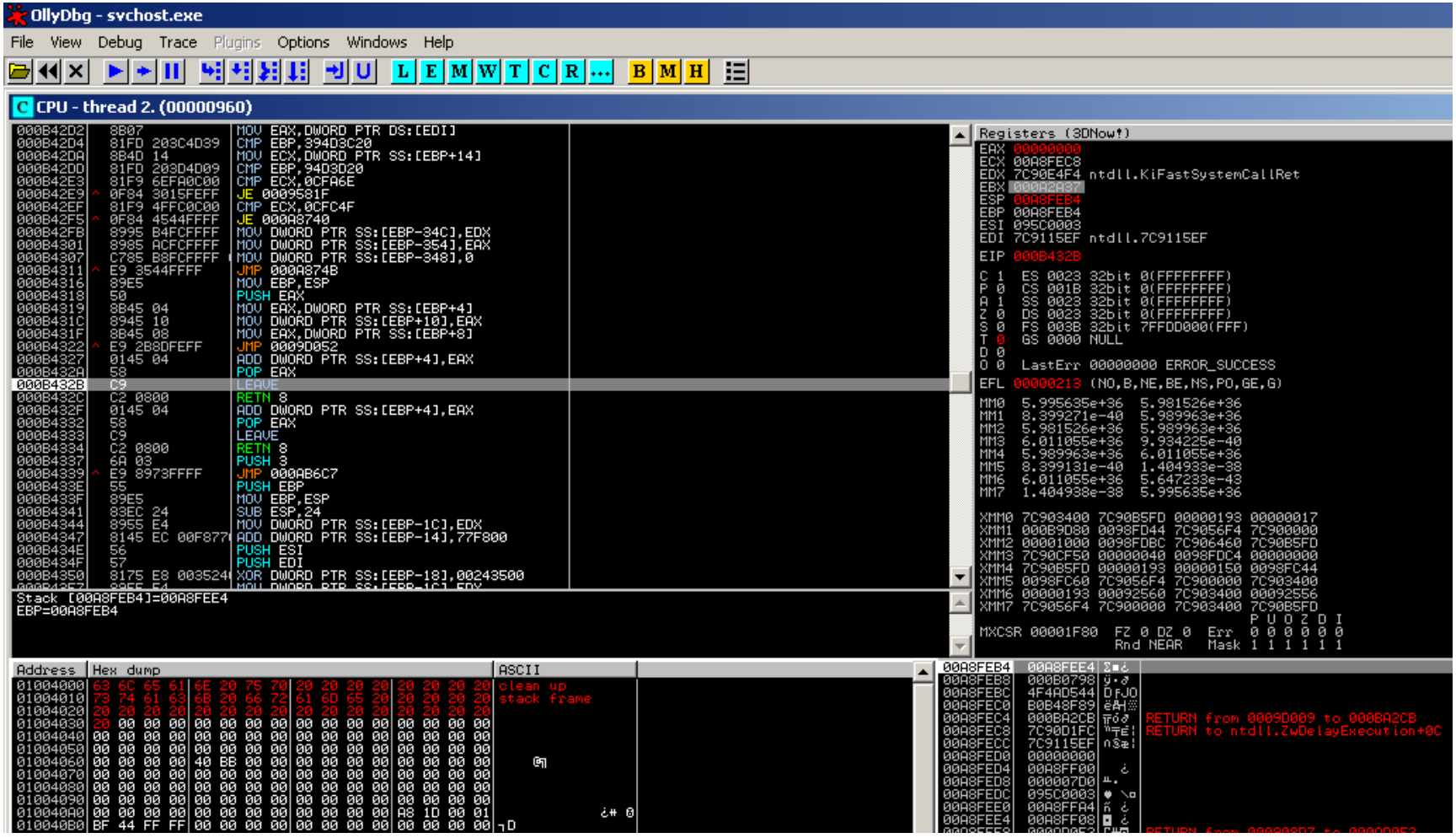
Patchwork

Example: Nymaim (Control Flow Obfuscation)



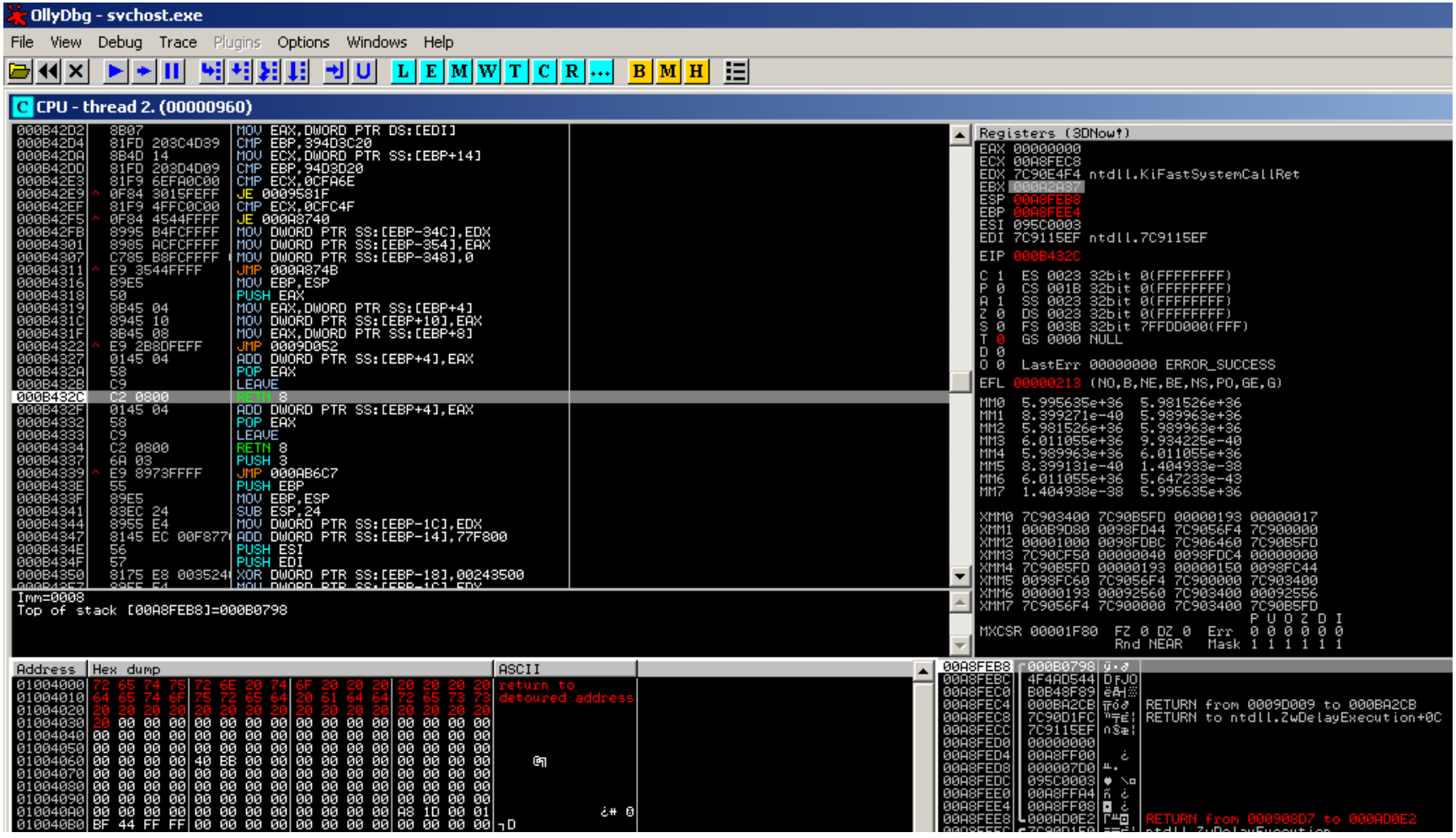
Patchwork

Example: Nymaim (Control Flow Obfuscation)



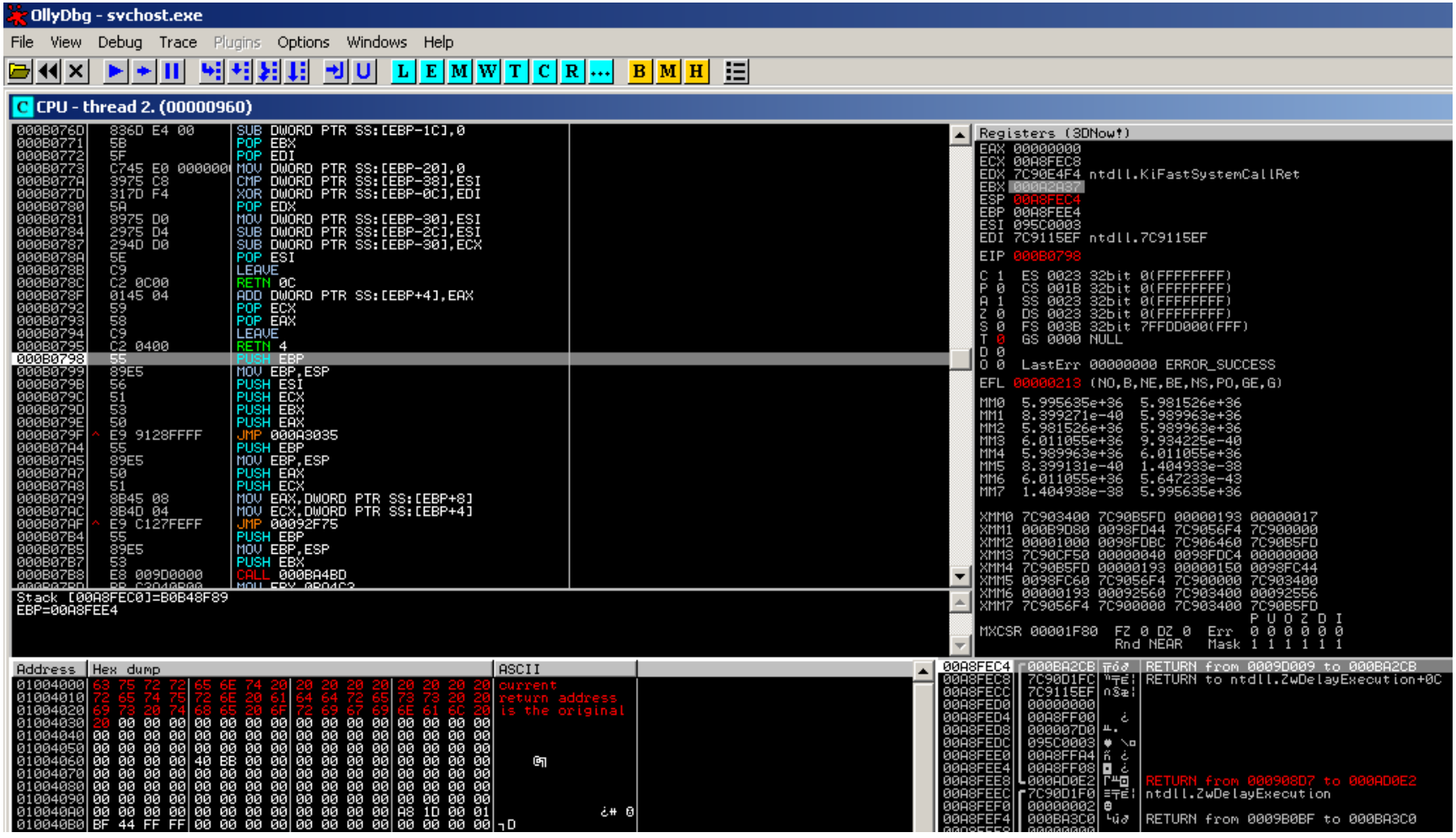
Patchwork

Example: Nymaim (Control Flow Obfuscation)



Patchwork

Example: Nymaim (Control Flow Obfuscation)



Patchwork
Example: Nymaim (Deobfuscation)

- Select:

```
50              push    eax
68 89 8F B4 B0  push    0B0B48F89h
68 44 D5 4A 4F  push    4F4AD544h
E8 3E 2D FE FF  call    sub_9D009
89 D8           mov     eax, ebx
```

```
push_push_call_regex = (
   r"\x68(?P<operand_1>[\S\s]{4})"
   r"\x68(?P<operand_2>[\S\s]{4})"
   r"\xE8"
)
```

- Emulate:

  Until first ret / retn instruction

Patchwork
Example: Nymaim (Deobfuscation)

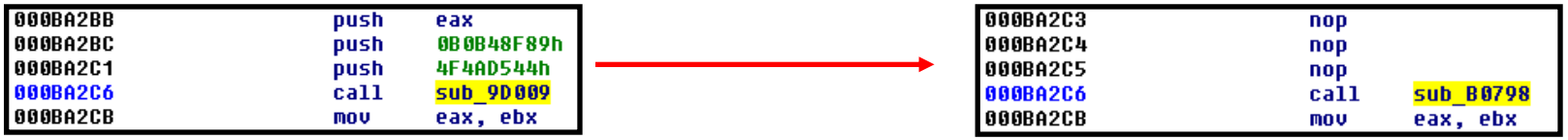
- Validate:

```
ppc_validators = {
   "call_detour": [
      'push dword',
      'push dword',
      'push ebp',
      'mov ebp,esp',
      'push eax',
      'mov eax,[ebp+0x4]',
      'mov [ebp+0x10],eax',
      'mov eax,[ebp+0xc]',
      '',  # contains the operand -> add, sub, xor
      'add [ebp+0x4],eax',
      'pop eax',
      'leave'],
```

Patchwork
Example: Nymaim (Deobfuscation)

- Transform:

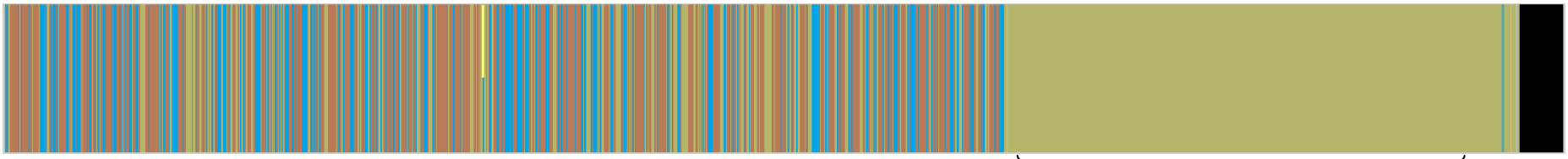
```
def _deobfuscate_call_detour(self, validation):
    obf_start_addr = validation.selection.selectionOffset
    call_offset = validation.emulation.cbResult - (obf_start_addr + 10 + 5)
    deobf_call = "\x90" * 10 + "\xE8" + struct.pack("I", (call_offset) & 0xffffffff)
    ida_lib.patch_bytes(obf_start_addr, deobf_call)
    self.updateCallXref(obf_start_addr + 10, validation.emulation.cbResult)
```



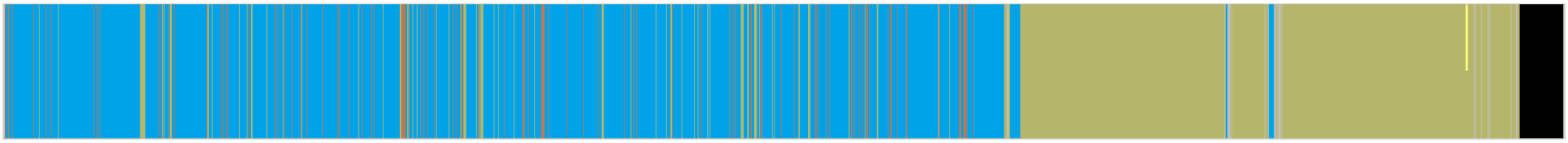
Patchwork
Example: Nymaim (Deobfuscation)

- Applying all deobfuscations:
  - ~2 min run time
  - 4443 transformations
  - Functions recognition: 463 -> 920

Before:



After:



Patchwork
Future plans

- Looking at more use cases
  - Memory usage analysis (deobfuscate Nymaim's blown up stack)
  - KINS BaseConfig (VM-based) decryption
  - Import reconstruction
- Extend / patch PyEmu
  - Change disassembly engine to IDA / capstone

    Increase coverage of opcodes

Patchwork
Conclusion

- Give it a try :)
  - Repository at http://patchwork.pnx.tf
    - (points to: https://bitbucket.org/daniel_plohmann/idapatchwork)
- Send feedback or ideas for improvement!
  - patchwork@pnx.tf / plohmann@cs.uni-bonn.de