Detecting Malicious Patterns in Executables via Model Checking

Mihai Christodorescu

mihai@cs.wisc.edu

University of Wisconsin, Madison

The Problem

- Malicious code is everywhere

☞ Viruses

  - Infect programs, cause damage

☞ Trojans & backdoors

  - Allow unauthorized remote access

☞ Spyware

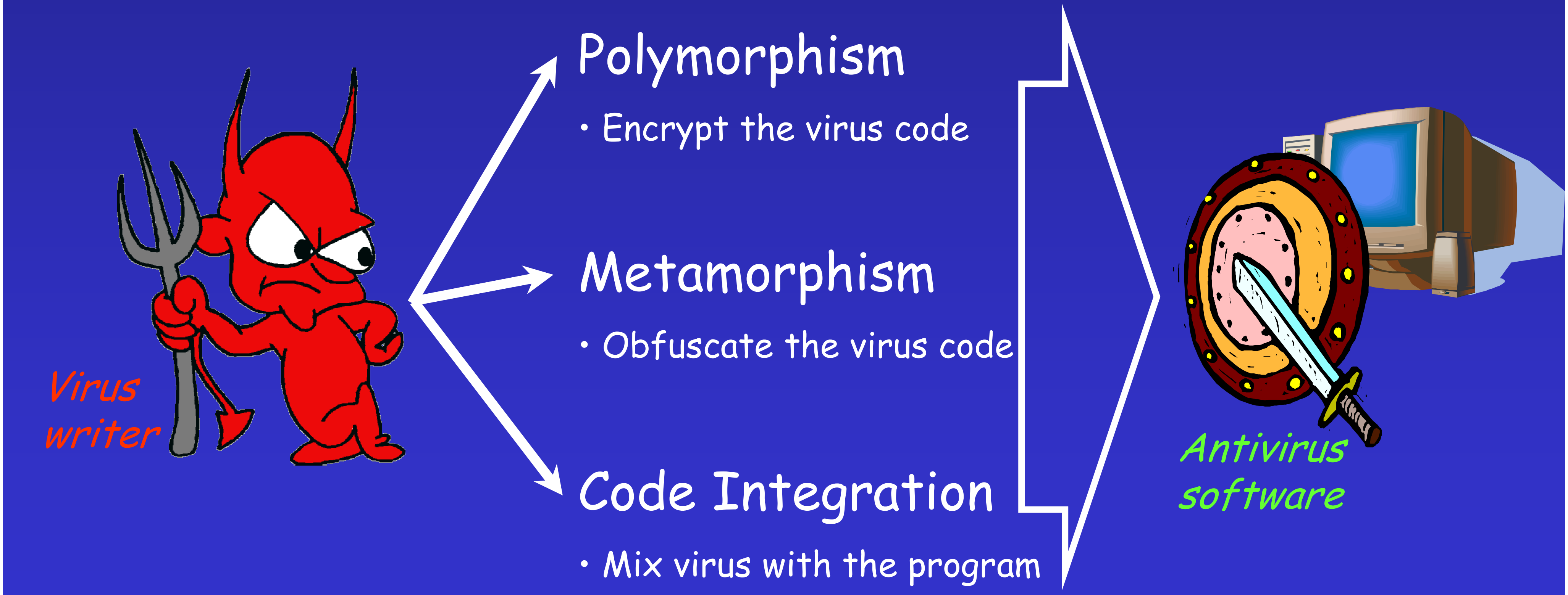
  - Monitor user activity, steal private data

☞ Worms

  - Move from machine to machine, through the network

Viruses

- Virus writers use complex techniques to obfuscate virus code



Polymorphism

- Encrypt the virus code

Metamorphism

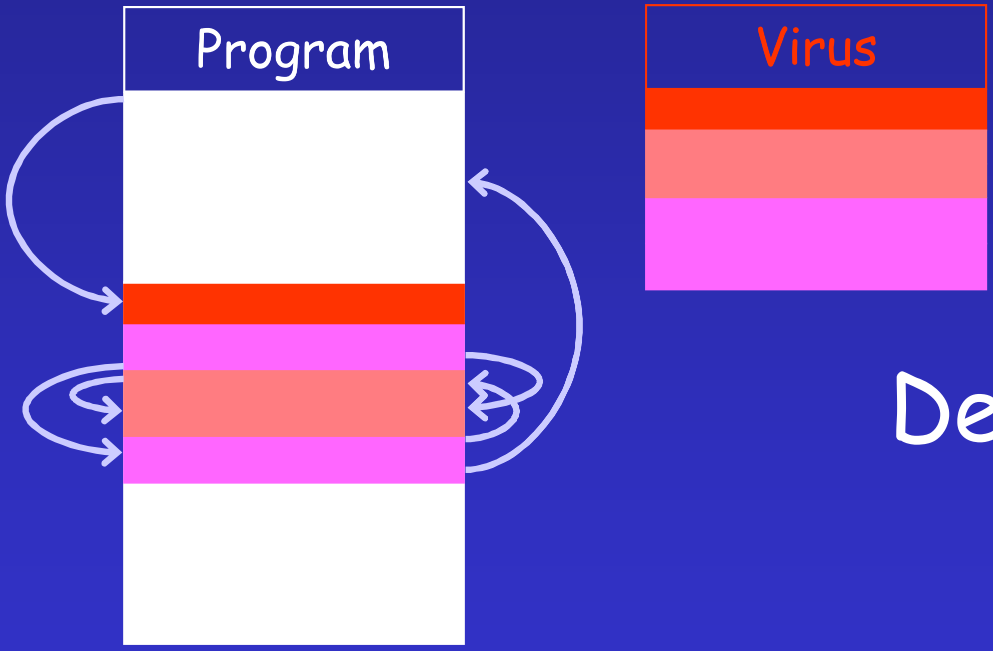
- Obfuscate the virus code

Code Integration

- Mix virus with the program

Obfuscation: <u>Metamorphism</u>

- Metamorphic viruses:
  - Morph the whole virus body

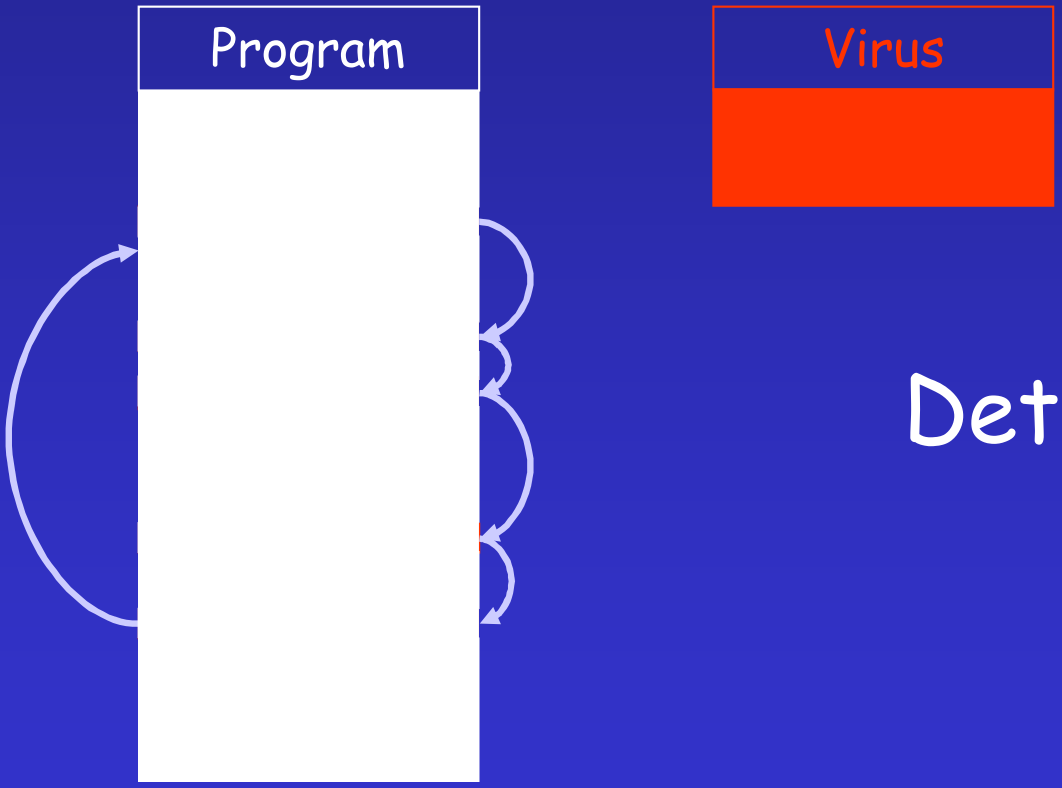


Detection methods:

?

Obfuscation: Code Integration

- Integration of virus and program
  - e.g. Mistfall Virus Engine



Detection methods:

?

Obfuscation Example

Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
          pop       ecx
          jecxz     SFModMark
          mov       esi, ecx
          mov       eax, 0d601h
          pop       edx
          pop       ecx
          call      edi
          jmp       Loop
```

Morphed Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
          pop       ecx
          nop
          jecxz     SFModMark
          xor       ebx, ebx
          beqz      N1
N1:       mov       esi, ecx
          nop
          mov       eax, 0d601h
          pop       edx
          pop       ecx
          nop
          call      edi
          xor       ebx, ebx
          beqz      N2
N2:       jmp       Loop
```

Obfuscation Example

Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
        pop      ecx
        jecxz    SFModMark
        mov      esi, ecx
        mov      eax, 0d601h
        pop      edx
        pop      ecx
        call     edi
        jmp      Loop
```

Morphed Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
        pop      ecx
        nop

        call     edi
        xor      ebx, ebx
        beqz     N2
N2:     jmp      Loop

        nop
        mov      eax, 0d601h
        pop      edx
        pop      ecx
        nop

        jecxz    SFModMark
        xor      ebx, ebx
        beqz     N1
N1:     mov      esi, ecx
```

Obfuscation Example

Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
          pop       ecx
          jecxz     SFModMark
          mov       esi, ecx
          mov       eax, 0d601h
          pop       edx
          pop       ecx
          call      edi
          jmp       Loop
```

Morphed Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
          pop       ecx
          nop
          jmp L1
L3:       call      edi
          xor       ebx, ebx
          beqz      N2
N2:       jmp       Loop
          jmp L4
L2:       nop
          mov       eax, 0d601h
          pop       edx
          pop       ecx
          nop
          jmp L3
L1:       jecxz     SFModMark
          xor       ebx, ebx
          beqz      N1
N1:       mov       esi, ecx
          jmp L2
L4:
```

Obfuscation Example

Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
            pop         ecx
            jecxz       SFModMark
            mov         esi, ecx
            mov         eax, 0d601h
            pop         edx
            pop         ecx
            call        edi
            jmp         Loop
```

Morphed Virus Code

(from Chernobyl CIH 1.4):

```
Loop:
            pop         ecx
            nop
            jmp L1
L3:         call        edi
            xor         ebx, ebx
            beqz        N2
N2:         jmp         Loop
            jmp L4
L2:         nop
            mov         eax, 0d601h
            pop         edx
            pop         ecx
            nop
            jmp L3
L1:         jecxz       SFModMark
            xor         ebx, ebx
            beqz        N1
N1:         mov         esi, ecx
            jmp L2
L4:
```

Current State of the Art

- Signature matching
  - Identify sequence of instructions unique to a virus

    => "virus signature"
    - Chernobyl signature: E800 0000 005B 8D4B 4251 5050
      0F01 4C24 FE5B 83C3 1CFA 8B2B
  - Scan programs for virus signature
  - Cumbersome, inaccurate

- Heuristics
  - Look for abnormal structures in certain program locations
    - Does the program start with a jump?
  - Inaccurate

Dismal State of the Art

Commercial antivirus tools vs. morphed known viruses

	Norton AntiVirus	COMMAND Software Systems
Chernobyl-1.4	× Not detected	× Not detected
f0sf0r0	× Not detected	× Not detected
Hare	× Not detected	× Not detected
z0mbie-6.b	× Not detected	× Not detected

What to do?

- Better code analysis tool
  - Analyze the program semantic structure (instead of signature or string matching)
    - Control flow
    - Data flow

- Check for presence of malicious properties
  - e.g.: "program writes to an executable file"
  - e.g.: "program monitors as executables are loaded into memory and changes them"
  - e.g.: "program behaves just like virus XYZ"

Overview

1. The Problem

☞ 2. Smart Virus Scanner

3. Results

4. Future Directions

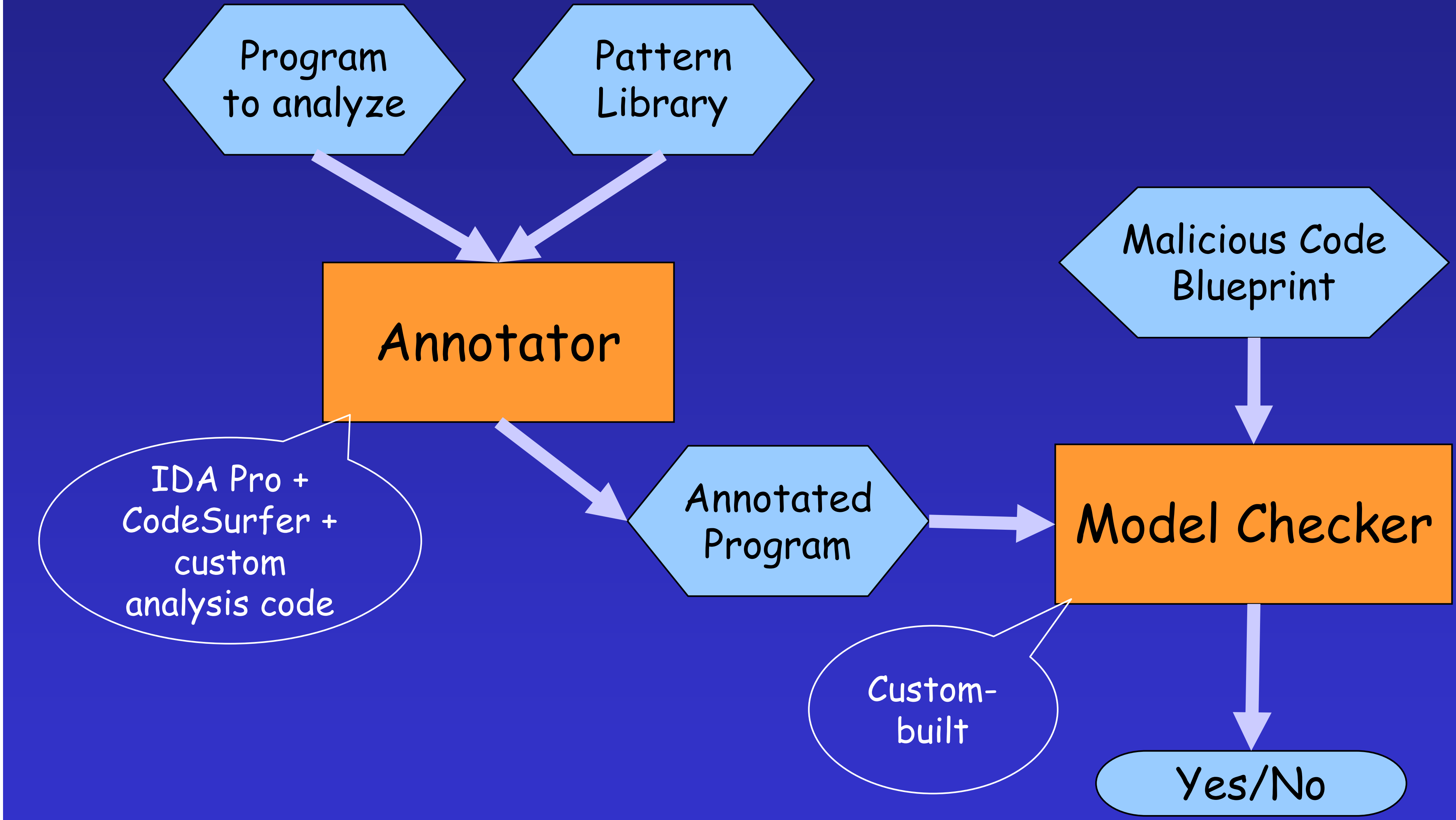
Smart Virus Scanner

1. Build automaton from vanilla virus
   - blueprint of malicious behavior

2. Build a model of the program

3. Check whether model “matches” the blueprint

Architecture



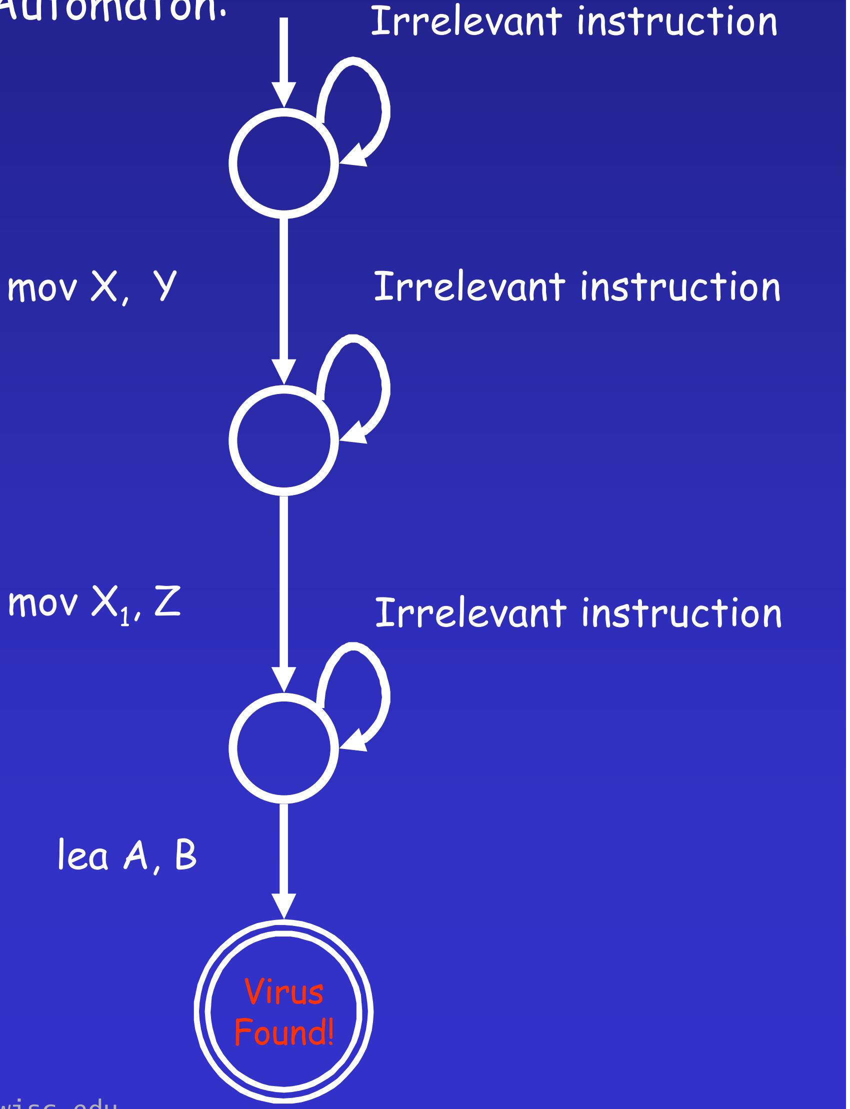
Detection Example

Virus Code:

```
push    eax
sidt    [esp-02h]
pop     ebx
add     ebx, HookNo * 08h + 04h
cli
mov     ebp, [ebx]
mov     bp, [ebx-04h]
lea     esi, MyHook - @1[ecx]
push    esi
mov     [ebx-04h], si
shr     esi, 16
mov     [ebx+02h], si
pop     esi
```

(from Chernobyl CIH 1.4 virus)

Virus Automaton:

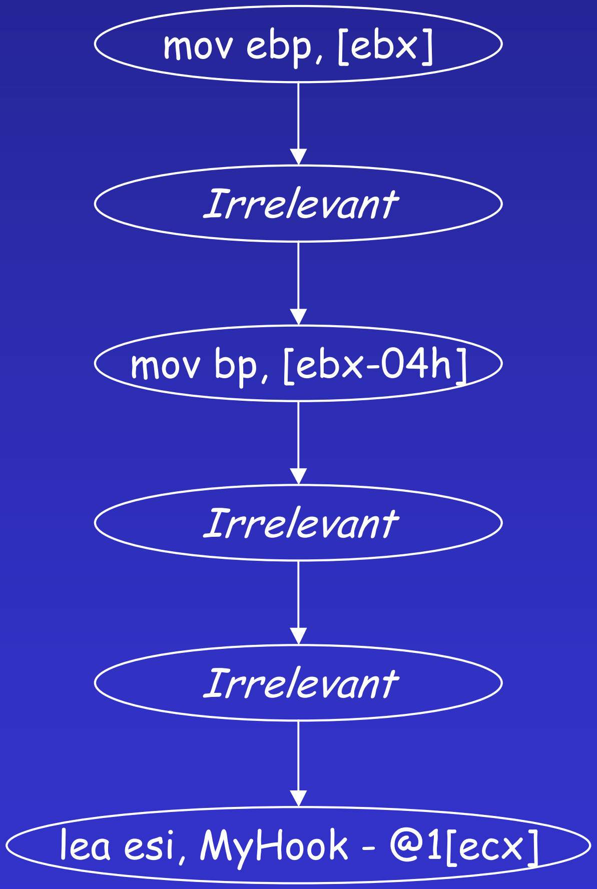


Detection Example

Program to be checked:

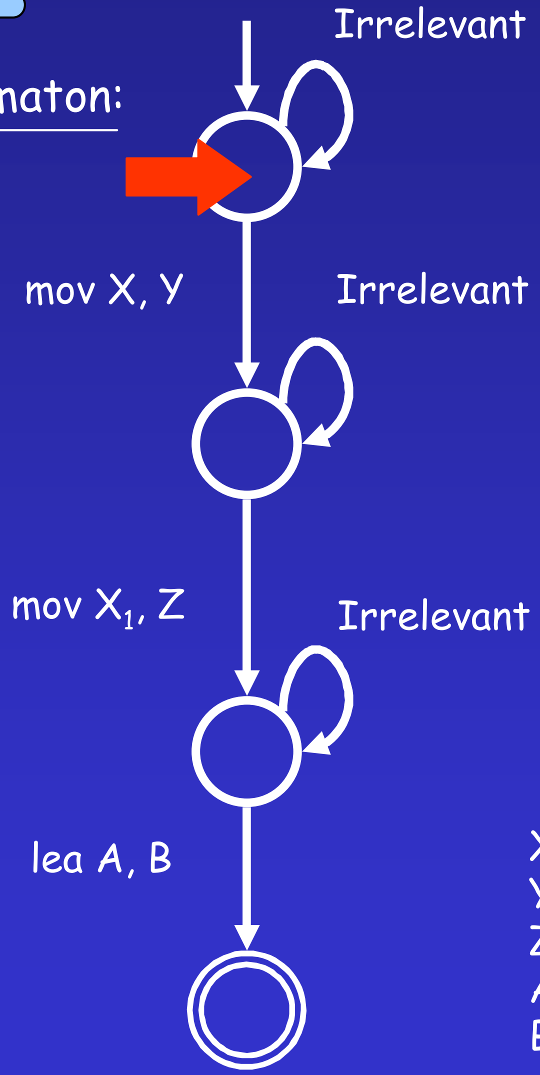
```
            mov ebp, [ebx]
            nop
            mov bp, [ebx-04h]
            test ebx
            beqz next
next:       lea esi, MyHook - @1[ecx]
```

Annotated program:

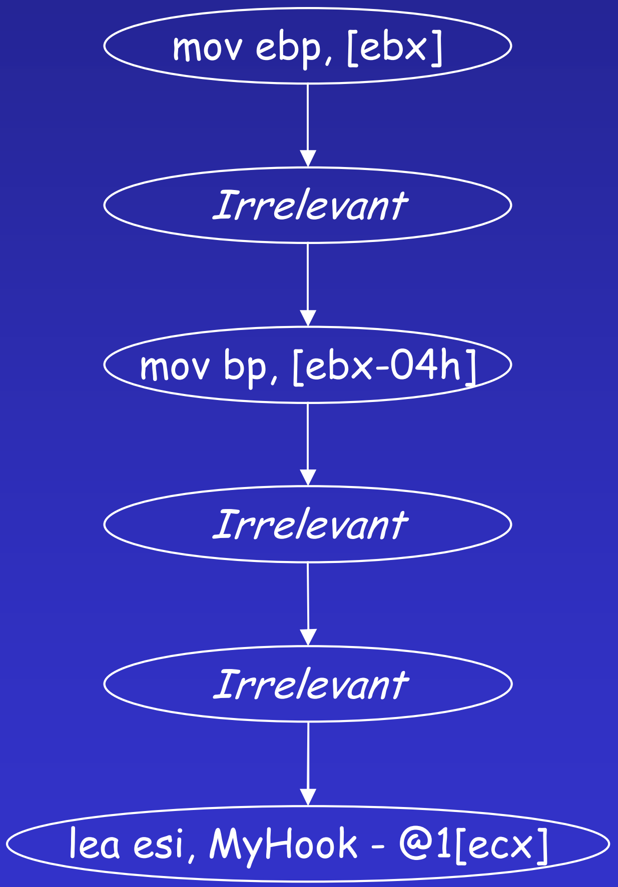


Detection Example

Virus Automaton:



Program model (annotated program):



X = ebp
Y = [ebx]
Z = [ebx – 04h]
A = esi
B = MyHook - @1[ecx]

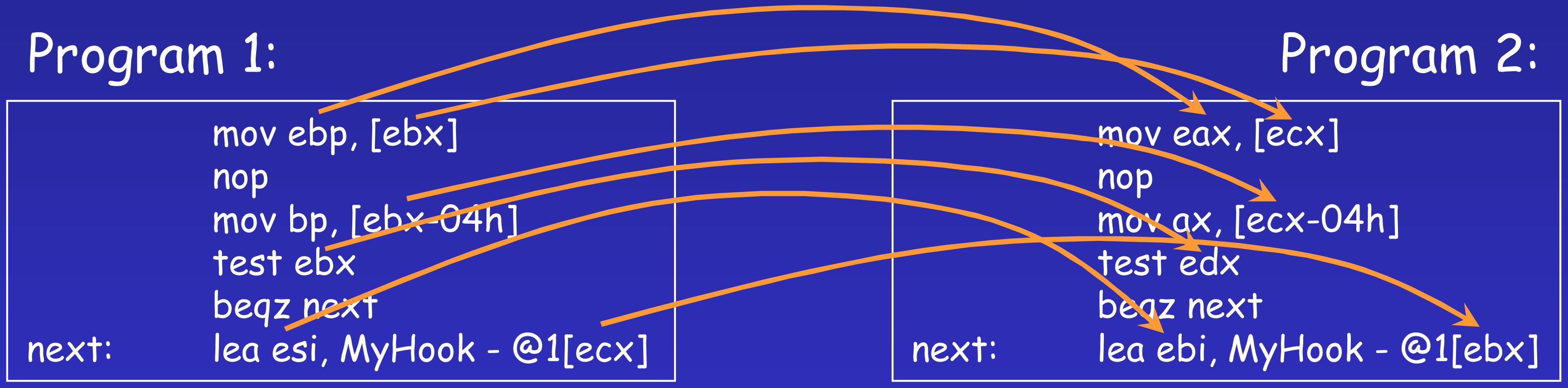
Smart Virus Scanner

- What are *irrelevant instructions*?
  - NOPs
  - Control flow instructions that do not change the control flow
    - e.g.: jumps/branches to the next instructions
  - Instructions that modify dead registers
  - Sequences of instructions that do not modify architectural state
    - e.g.:

```
add ebx, 1
sub ebx, 1
```

Uninterpreted Symbols

- What happens when the registers are changed?



Program 1:

```
          mov ebp, [ebx]
          nop
          mov bp, [ebx-04h]
          test ebx
          beqz next
next:     lea esi, MyHook - @1[ecx]
```

Program 2:

```
          mov eax, [ecx]
          nop
          mov ax, [ecx-04h]
          test edx
          beqz next
next:     lea ebi, MyHook - @1[ebx]
```

Virus Spec:

```
          mov ebp, [ebx]
```

=> No match with Program 2

Virus Spec with *Uninterpreted Symbols*:

```
          mov X, Y
```

=> Matches both Programs 1 and 2

Overview

1. The Problem

2. Smart Virus Scanner

☞ 3. Results

4. Future Directions

Results

- Testing
  - Viruses used: Chernobyl, Hare, z0mbie-6.b, f0sf0r0
  - Antivirus utilities
    - Command AntiVirus (F-Prot)
    - Norton AntiVirus (Symantec)

☹ Not surprising!

- Norton and Command AV do not detect morphed viruses

☺ Our Smart Virus Scanner catches morphed viruses

Results

- The detection tool can handle:
  - NOP-insertion
  - Code reordering
  - Irrelevant jumps and branches
  - Irrelevant procedure calls
  - Register renaming

- Work in progress:
  - Inter-procedural analysis
  - Extended irrelevant code detection

Implementation Status

- Annotator - completed
- Model Checker – completed
  (first version)
- Features
  - Modular
    - Relatively easy to analyze different types of executable code
  - Extensible
    - New static analyses can be added to enhance the malicious code detection

Overview

1. The Problem
2. Smart Virus Scanner
3. Results
4. ☞ Future Directions

Future Directions

- New formats/languages
  - scripts (Visual Basic, ASP, Javascript)
  - multi-language malicious code

- Attack diversity
  - beyond viruses:
    - trojans/backdoors
    - spyware
    - worms

Future Directions

- Better static analyses
  - Polyhedral analysis
  - Pointer analysis
    - fundamental for interprocedural algorithms
    - necessary for Intel/x86-like (CISC) platforms

- Short term
  - Refine and optimize current toolkit

References

- Schneider, F.B. *Enforceable Security Policies*. TR99-1759, July 27, 1999.

- Dawson Engler, Benjamin Chelf, Andy Chou, and Seth Hallem. *Checking System Rules Using System Specific, Programmer-Written Compiler Extensions*. In Proceedings of the 4th OSDI Symposium, San Diego, CA, October 2000. http://citeseer.nj.nec.com/engler00checking.html

- Péter Ször, and Peter Ferrie. *Hunting For Metamorphic*. In Proceedings of Virus Bulletin Conference, September 2001. Pp. 123 – 154.
  http://www.geocities.com/szorp/metamorp.pdf

- Zombie. *Zombie's Homepage*.
  http://z0mbie.host.sk

- Usenet: alt.comp.virus.source.code

Conclusions

- Better program analysis technique leads to more malicious code detection power
- Modular design will allow for analysis of both assembly and scripting languages