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: **Metamorphism**

- **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:

<table>
<thead>
<tr>
<th>pop ecx</th>
<th>nop</th>
<th>jecxz SFModMark</th>
<th>xor ebx, ebx</th>
<th>beqz N1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1: mov esi, ecx</td>
<td>nop</td>
<td>mov eax, 0d601h</td>
<td>pop edx</td>
<td>pop ecx</td>
</tr>
<tr>
<td>N2:</td>
<td>nop</td>
<td>call edi</td>
<td>xor ebx, ebx</td>
<td>beqz N2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jmp Loop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Obfuscation Example**

<table>
<thead>
<tr>
<th>Virus Code</th>
<th>Morphed Virus Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(from Chernobyl CIH 1.4):</strong></td>
<td><strong>(from Chernobyl CIH 1.4):</strong></td>
</tr>
<tr>
<td>Loop:</td>
<td>Loop:</td>
</tr>
<tr>
<td>pop</td>
<td>pop</td>
</tr>
<tr>
<td>jecxz</td>
<td>nop</td>
</tr>
<tr>
<td>mov</td>
<td>call</td>
</tr>
<tr>
<td>mov</td>
<td>xor</td>
</tr>
<tr>
<td>ebx</td>
<td>beqz</td>
</tr>
<tr>
<td>ebx</td>
<td>mov</td>
</tr>
<tr>
<td>ecx</td>
<td>pop</td>
</tr>
<tr>
<td>ecx</td>
<td>pop</td>
</tr>
<tr>
<td>edi</td>
<td>pop</td>
</tr>
<tr>
<td>Loop</td>
<td>nop</td>
</tr>
<tr>
<td>sms</td>
<td>jecxz</td>
</tr>
<tr>
<td>ebx</td>
<td>beqz</td>
</tr>
<tr>
<td>ebx</td>
<td>mov</td>
</tr>
</tbody>
</table>
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
jmp L4

L2:
nop
mov eax, 0d601h
pop edx
pop ecx
nop
jmp L3

L1:
jecxz SFModMark
xor ebx, ebx
beqz N1
mov esi, ecx
jmp L2

L4:
## Obfuscation Example

### Virus Code
*(from Chernobyl CIH 1.4):*

<table>
<thead>
<tr>
<th>Loop</th>
<th>pop</th>
<th>ecx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>jecxz</td>
<td>SFModMark</td>
</tr>
<tr>
<td></td>
<td>mov</td>
<td>esi, ecx</td>
</tr>
<tr>
<td></td>
<td>mov</td>
<td>eax, 0d601h</td>
</tr>
<tr>
<td></td>
<td>pop</td>
<td>edx</td>
</tr>
<tr>
<td></td>
<td>pop</td>
<td>ecx</td>
</tr>
<tr>
<td></td>
<td>call</td>
<td>edi</td>
</tr>
<tr>
<td></td>
<td>jmp</td>
<td>Loop</td>
</tr>
</tbody>
</table>

### Morphed Virus Code
*(from Chernobyl CIH 1.4):*

<table>
<thead>
<tr>
<th>Loop</th>
<th>pop</th>
<th>ecx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nop</td>
<td>ecx</td>
</tr>
<tr>
<td></td>
<td>jmp</td>
<td>L1</td>
</tr>
<tr>
<td>L3:</td>
<td>call</td>
<td>edi</td>
</tr>
<tr>
<td></td>
<td>xor</td>
<td>ebx, ebx</td>
</tr>
<tr>
<td></td>
<td>beqz</td>
<td>N2</td>
</tr>
<tr>
<td>N2:</td>
<td>jmp</td>
<td>L4</td>
</tr>
<tr>
<td></td>
<td>jmp</td>
<td>Loop</td>
</tr>
<tr>
<td>L2:</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mov</td>
<td>eax, 0d601h</td>
</tr>
<tr>
<td></td>
<td>pop</td>
<td>edx</td>
</tr>
<tr>
<td></td>
<td>pop</td>
<td>ecx</td>
</tr>
<tr>
<td></td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>jmp</td>
<td>L3</td>
</tr>
<tr>
<td>L1:</td>
<td>jecxz</td>
<td>SFModMark</td>
</tr>
<tr>
<td></td>
<td>xor</td>
<td>ebx, ebx</td>
</tr>
<tr>
<td></td>
<td>beqz</td>
<td>N1</td>
</tr>
<tr>
<td>N1:</td>
<td>mov</td>
<td>esi, ecx</td>
</tr>
<tr>
<td></td>
<td>jmp</td>
<td>L2</td>
</tr>
<tr>
<td>L4:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Current State of the Art

• Signature matching
  - Identify sequence of instructions unique to a virus
    \Rightarrow \text{“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  
|------------------|------------------|------------------
| 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

Program to analyze

Pattern Library

Annotator

Malicious Code Blueprint

Annotated Program

Model Checker

IDA Pro + CodeSurfer + custom analysis code

Custom-built

Yes/No
Detection Example

Virus Code:

```
push    eax
sidt    [esp-02h]
pop     ebx
add     ebx, HookNo * 08h + 04h
cl	
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:

```
mov ebp, [ebx]

Irrelevant

mov bp, [ebx-04h]

Irrelevant

Irrelevant

lea esi, MyHook - @1[ecx]
```
Detection Example

Virus Automaton:

- `mov X, Y`
- `mov X1, Z`
- `lea A, B`

Program model (annotated program):

- `mov ebp, [ebx]`
- `lea esi, MyHook - @1[ecx]`
- `mov bp, [ebx-04h]`
- `lea esi, MyHook - @1[ecx]`
- `irelevant`
- `irelevant`
- `irelevant`
- `irelevant`
- `irelevant`

Equations:

- `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`
- `lea esi, MyHook - @1[ecx]`
- `next:

Program 2:
- `mov eax, [ecx]`
- `nop`
- `mov ax, [ecx-04h]`
- `test edx beqz next`
- `lea ebi, MyHook - @1[ebx]`
- `next:

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


• Zombie. *Zombie’s Homepage*. [http://z0mbie.host.sk](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