Model Checking for Binaries

Mihai Christodorescu
University of Wisconsin
Virus Scanning

• Viruses
  - Likely to become a bigger problem in the future

• Virus scanners
  - Check binaries for each type of virus
  - Virus signatures
Virus Scanners

Scanning

Executable

match?
match?
match?

Clean/Infected

Virus Scanning Engine

Signature for Virus A
Signature for Virus B
Signature for Virus C

Update

Update

New Virus D

Signature for Virus D
Virus Scanner Limitations

- Signatures are sequences of instructions
  - Small changes to infected program do not trigger a match

Virus Signature

mov [eax + OCh], ebx
mul edx, [ebp + FFh]
add ebp, 2

Infected Program

mov [eax + OCh], ebx
nop
mul edx, [ebp + FFh]
test ebx
beqz next
next:
add ebp, 2
Is It Really A Problem?

• Yes!
  - Virus writers are getting smarter
  - Viruses are better at hiding themselves
What to do?

• Build a better mouse trap!
• Better detection tool
  - Use static analysis information
  - More flexible

=> Smart Virus Scanner

Demo 1  Demo 2  Demo 3
Smart Virus Scanner

1. Build a model of the virus code
   - Blueprint of virus behavior

2. Verify that program does not match the blueprint

• More powerful
Smart Virus Scanner Example

**Virus Signature:**

```
mov [eax + 0Ch], ebx
mul edx, [ebp + FFh]
add ebp, 2
```

**Virus Model:**

```
mov [eax + 0Ch], ebx
mul edx, [ebp + FFh]
add ebp, 2
```

Irrelevant instruction

Irrelevant instruction

Irrelevant instruction

Irrelevant instruction
Smart Virus Scanner Example

Virus Model:

Irrelevant instruction

mov [eax + 0Ch], ebx

Irrelevant instruction

mul edx, [ebp + FFh]

Irrelevant instruction

add ebp, 2

Program to be checked:

mov [eax + 0Ch], ebx
nop
mul edx, [ebp + FFh]
test ebx
beqz next
next: add ebp, 2
Model Checking

- Technique for checking program properties
- Build a “model program” that obeys the property
- Compare the program against the model
Model Checking

Security Automaton proposed by [Schneider 1999].
Model Checking

• Security Automaton
  - For policy “Always release a resource after acquiring it.”

![Diagram of Security Automaton]

- States: $S_0$, $S_1$
- Transitions:
  - Acquire
  - Release
  - Other

11/08/2001 Mihai Christodorescu - mihai@cs.wisc.edu
Model Checking

• Abstract Representation

```
x = read_input()
y = 0
z = 2 * PI
result = undefined
```

```
x > 0 ?
result = compute( x, y, z )
z = 0
write_output( result )
```

```
x = read_input()
```

```
compute( x, y, z )
```

```
write_output( result )
```
Model Checking for Binaries

Abstract Representation

Security Automaton for P

OK/Failed

ID A Pro (+ SDK plugins)

Transducer (on top of CodeSurfer static analysis engine)

Model Checker
Current Status

• Transducer
  - Works on sequences of instructions (basic blocks) to build the abstract representation
  - Uses only the static analysis information provided by CodeSurfer
Current Status

• Model Checker
  - Intra-procedural only

Procedure P

Procedure Q

Not used!
Static Analysis and Model Checking

• More (precise) information from static analysis can enhance model checking

• Smart Virus Scanner
  - “irrelevant” instructions - better identified by alias/points-to analysis
Future Directions

• Context-sensitive model checking

• Better static analyses

• Model checking for component systems
References
