# Access Control Based on Dynamic Monitoring for Detecting Software Malicious Behaviours

*K. Adi, L. Sullivan & A. El Kabbal*

Computer Security Research Laboratory

http://w3.uqo.ca/lrsi

# Motivation

## Malicious software (or malware)

- software specifically designed to perform unauthorized actions to extract information from or damage a computing device.

## Common classes of malware include

- Viruses, Worms and Trojan horses
- Adware, Spyware
- Bots
- Phishing

## Controlling malware is an access control issue

- **Challenge**: how to identify and block their access

# A short taxonomy of malware detection methods

## Signature based detection

- Scans a disk for characteristic content of known malware
  - (+) 100% detection rate
  - (-) Detects only known malware

## CRC checker

- Verifies whether files on disk have been modified
  - (+) 100% detection rate
  - (-) Detection is after the fact
  - (-) False positives

# A short taxonomy (cont'd)

## Heuristic based detection

- Scanner examines overall code structure for signs of malicious logic.
  - (+) Detection before infection
  - (-) Easy to hide from the scanner
  - (-) False positive rates can be high

# Behaviour Based Detection

- Detects and blocks malware based on specified behaviour.
- Some example behaviour could include:
  - Attempts to format a disk drive
  - Modification of start-up settings
  - Initiation of a network communication
  - Scripting of email to send executable files

(+) detects unknown malware

(+) not dependent on updates

(-) may cause false positives

# No perfect panacea

## Most everyone agrees

- No one method is perfect: layered approach to detection is probably best

## More Control to user

- Behaviour based methods gives user some control over what type of behaviour he or she will tolerate.

# Basic architecture of a Behavior Based Malware Detection System

1. Choose characteristics upon which to base behaviour detection
   - OS calls
   - Access to specific files
   - Access to registry keys
   - Access to communication ports
   - etc.
2. Collect behaviour data from running software application
3. Analyse collected data to determine if behaviour is deemed malicious
   - If so, block the application from executing further

# Virimon: A malware detector and blocker

## Characteristic chosen

- OS calls

## Data collection

- Kernel level driver collects OS call information from running target application

## Analysis and reaction

- Uses model-checking to detect malicious behaviour
- Driver can block further execution

# Model Checking

## Formal methods technique

- Verifies whether a model of a system satisfies some specified claim.

## Build a finite state model M of the system

- Usually requires some simplification of the system

## Specify a claim $\sigma$ using a chosen logic

## Model-Checking algorithm

- Check that the model M satisfies $\sigma$

# Virimon: modeling the system

Modeling single execution of a software application

- Modeled as a sequence of OS calls

Represent model as a Finite State Automaton (FSA)

Each state $s_i$ is a set of OS calls made

- A transition $(s_i, l_i, s_{i+1})$ represents a particular OS call

The model is a possibly infinite set of transitions

# Example : a malicious text editor

## Sequence of OS calls made by a text editor

- i*th* call : open port
- j*th* call : open file
- k*th* call: write on port

  with i < j < k

## Resulting FSA

# Virimon: Specifying the malicious behaviour

## Linear Temporal Logic (LTL)

- Allows specification of a malicious behaviour in terms of temporal events

## Well-formed LTL formula

- built from state formula and temporal operators

## LTL claim for the text editor example

$$\Diamond(openfile == true) \quad o \quad \Diamond(writeonport == true)$$

- “An OpenFile command is followed by the execution of a WriteOnPort command”

# LTL and FSA's

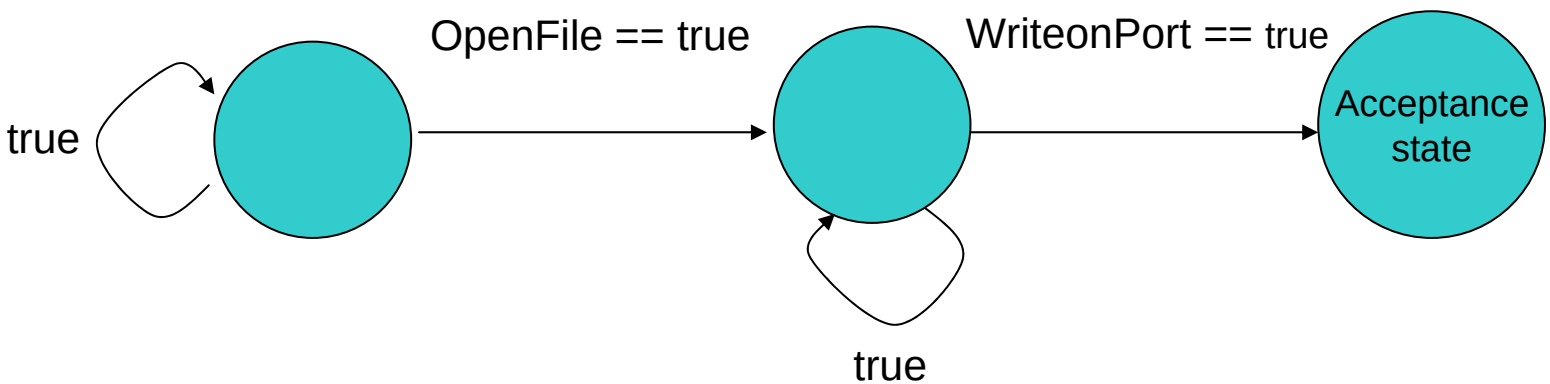
## LTL used

- To specify claims that represents malicious behaviour

## LTL claim

- can be converted to an FSA

## In our example, the claim becomes



# Comparing the model against the malicious behaviour

Form the intersection of the ‘languages’

- Of the model’s FSA and
- Of the FSA of the malicious behaviour claim

Determine if the intersection is empty or not

- if non-empty the claim holds

# Virimon: behavior based malware detector and blocker

## Two main components

- Kernel level driver to intercept OS calls from target application
- Application level verifier implementing a model-checker (based on the SPIN project*)

*G. Holzmann. The model Checker SPIN. Software Engineering, 23(5):279-295,1997.

# Virimon driver: Driver OS call extractor

## Kernel level driver

- Continually intercepts OS calls made by a targeted application
- Transmits each intercepted OS call to the verification module

## Verification module

- Builds an execution trace of the targeted application
- Checks the LTL formula against the execution trace
- Depending on the result of model-checking
  - Allows target application to proceed or
  - Blocks its execution

# Implementation details & improvements

LTL is a rich language

- Can be used to specify almost any desired property.

29 OS calls can be specified in Virimon, including (more to come)

- Openfile, WriteFile, ReadFile, CreateFile, QueryInformationFile, ...
  QuerySystemInformation, QuerySymbolicLinkObject, ...
  Createkey , Openkey, Deletekey ,EnumerateKey,...

Parameters are not yet specifiable

- To be implemented in later version.

Improve speed of model-checker

- Reduce waiting time while application is paused

Reduce false alarms

- Research required

# Other possible uses of Virimon

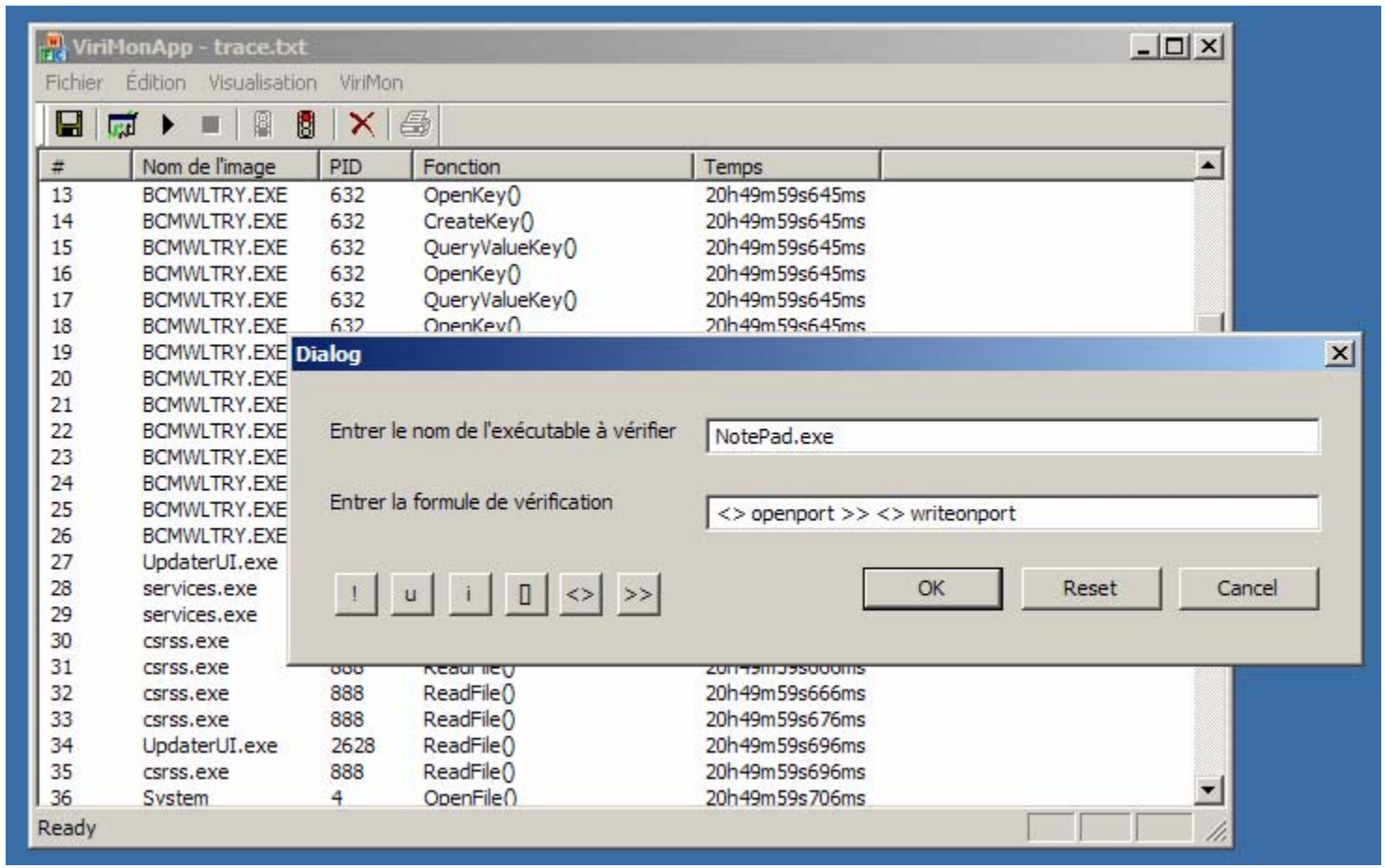
## Software Testing

- Allows tester to discover if a certain behaviour is present in a software application

## Software understanding

- Virimon will  list the OS calls produced by the target application.

# Prototype


ViriMonApp - trace.txt
Fichier Édition Visualisation ViriMon
# Nom de l'image PID Fonction Temps
13 BCMWLTRY.EXE 632 OpenKey() 20h49m59s645ms
14 BCMWLTRY.EXE 632 CreateKey() 20h49m59s645ms
15 BCMWLTRY.EXE 632 QueryValueKey() 20h49m59s645ms
16 BCMWLTRY.EXE 632 OpenKey() 20h49m59s645ms
17 BCMWLTRY.EXE 632 QueryValueKey() 20h49m59s645ms
Dialog
Entrer le nom de l'exécutable à vérifier
NotePad.exe
Entrer la formule de vérification
<> openport >> <> writeonport
! u i [] <> >>
OK Reset Cancel
27 UpdaterUI.exe
28 services.exe
29 services.exe
30 csrss.exe
31 csrss.exe
32 csrss.exe 888 ReadFile() 20h49m59s666ms
33 csrss.exe 888 ReadFile() 20h49m59s676ms
34 UpdaterUI.exe 2628 ReadFile() 20h49m59s696ms
35 csrss.exe 888 ReadFile() 20h49m59s696ms
36 System 4 OpenFile() 20h49m59s706ms
Ready

# Conclusion

Virimon is a behaviour based malware detector and blocker

- Powerful technique for malware detection
- Virimon is implemented in both kernel and application levels to control access

Formal methods in Virimon

- Allows  a rich semantics for specifying malicious behaviour
- Guarantees that malicious behaviour will be found

A working prototype gives very promising results