Malware Analysis IV

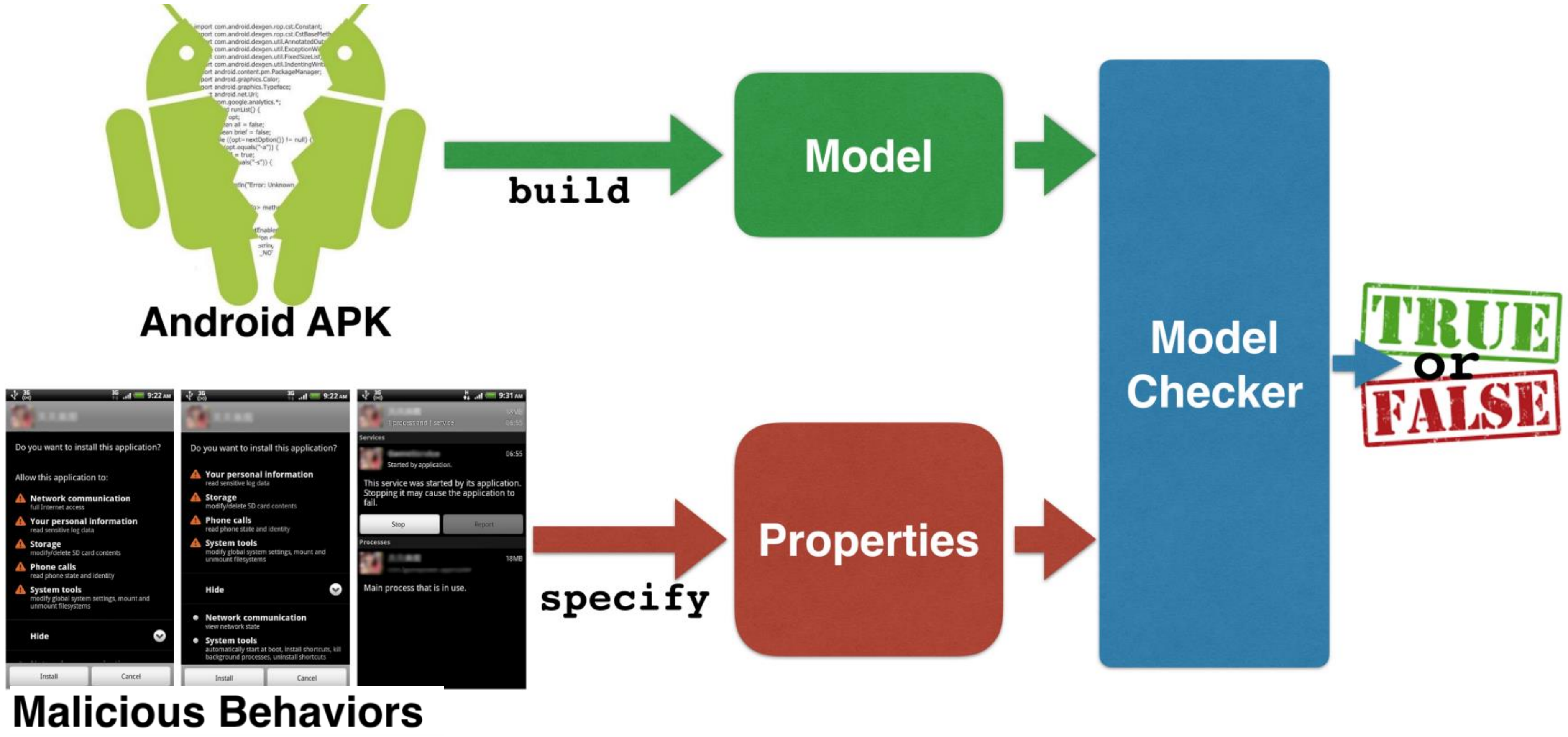
Francesco Mercaldo
University of Molise, IIT-CNR
francesco.mercaldo@unimol.it

Formal Methods for Secure Systems, University of Pisa - 17/04/2020

Finding Malicious Behaviour

- The manual ispection process
- Take a look to the Bytecode and the Java source code

A Formal Framework for Mobile Malicious Detection



Do you want to install this application?

Allow this application to:

- Network communication — full Internet access
- Your personal information — read sensitive log data
- Storage — modify/delete SD card contents
- Phone calls — read phone state and identity
- System tools — modify global system settings, mount and unmount filesystems

Hide

Install | Cancel

Do you want to install this application?

- Your personal information — read sensitive log data
- Storage — modify/delete SD card contents
- Phone calls — read phone state and identity
- System tools — modify global system settings, mount and unmount filesystems

Hide

- Network communication — view network state
- System tools — automatically start at boot, install shortcuts, kill background processes, uninstall shortcuts

Install | Cancel

1 process and 1 service

Services

06:55 Started by application.

This service was started by its application. Stopping it may cause the application to fail.

Stop | Report

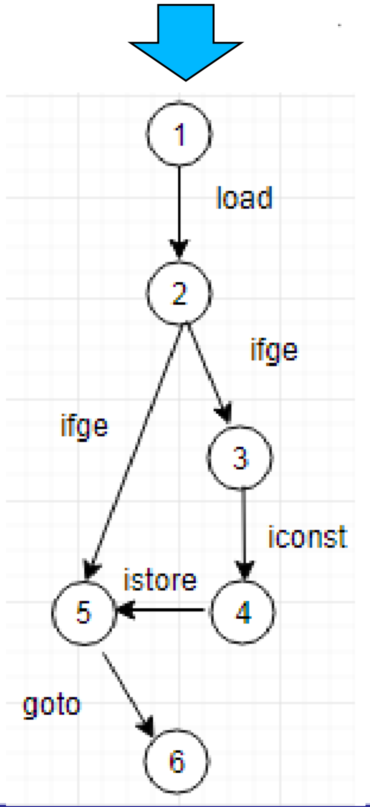
Processes

18MB

Main process that is in use.

From bytecode to automaton

```
.....
1:  load x                  // load x on the stack
2:  ifge 5                  // jump if greater or equal
3: iconst_1                 // push 1 on the stack
4: istore_2                 // pop off the stack and store the value in a register
5: goto 6                   // jump
6: .....
```



Why Formal Methods?

- The checking process is automatic, there is no need to construct a correctness proof
- The possibility of using the diagnostic counterexamples
- Temporal logic can easily and correctly express the behaviour of a spyware application
- Formal verification allows evaluating all possible scenarios, the entire state space all at once

Model checking for malware analysis

- Model: control flow graph of the code (plus some additional information)
- Logic formula: malware pattern

Verification by model checking algorithms that the model satisfies the formula

The verification is automated

Many model checking tools exist, with different formalisms for the specification of the model and for the specification of the properties.

- NOME : LEILA
- Developed at Univ. of Molise, Campobasso /IIT-CNR, Pisa
- Malware detection using model checking in android app
- Based on the CWB model checker (CCS and mu-calculus)

Example: Sequences of bytecode instructions in the CFG of a method

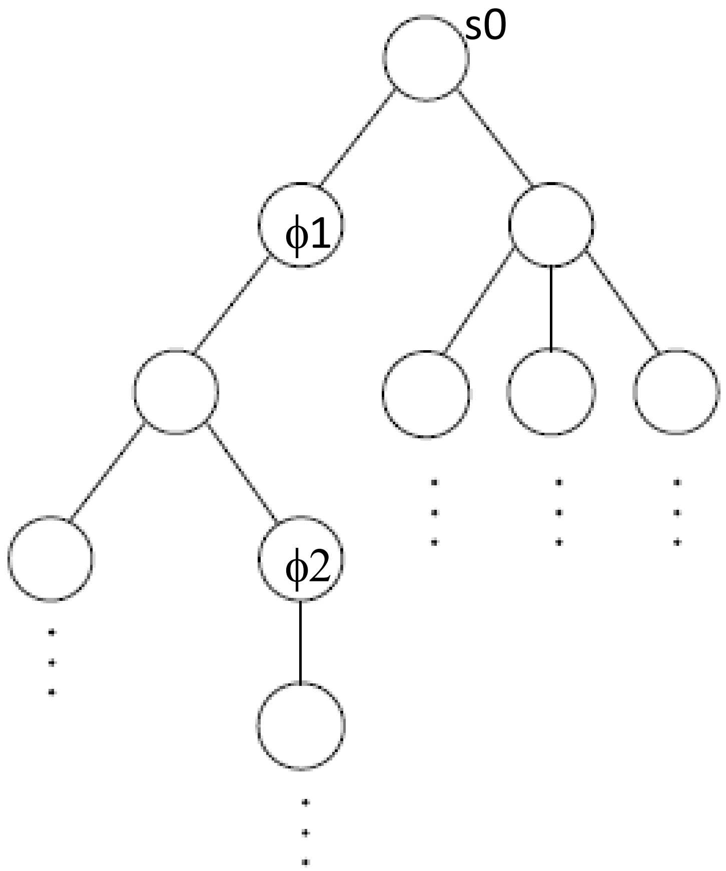
ϕ1: ldc *Activation*

ϕ2: ldc *HomePage*

Property 1 (P1)

There exists a path such that: a state is reached that satisfies ϕ1 and successively a state is reached that satisfies ϕ2

S0 |= P1 TRUE



Example: Sequences of bytecode instructions in the CFG of a method

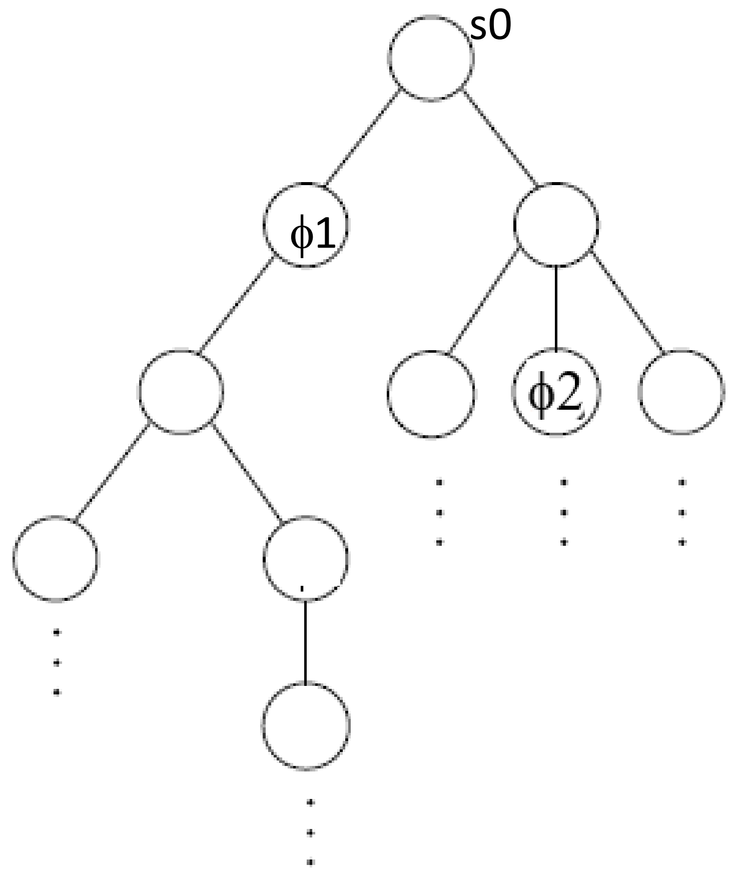
ϕ1: ldc *Activation*

ϕ2: ldc *HomePage*

Property 1 (P1)

There exists a path such that: a state is reached that satisfies ϕ1 and successively a state is reached that satisfies ϕ2

S0 |= P1 TRUE?



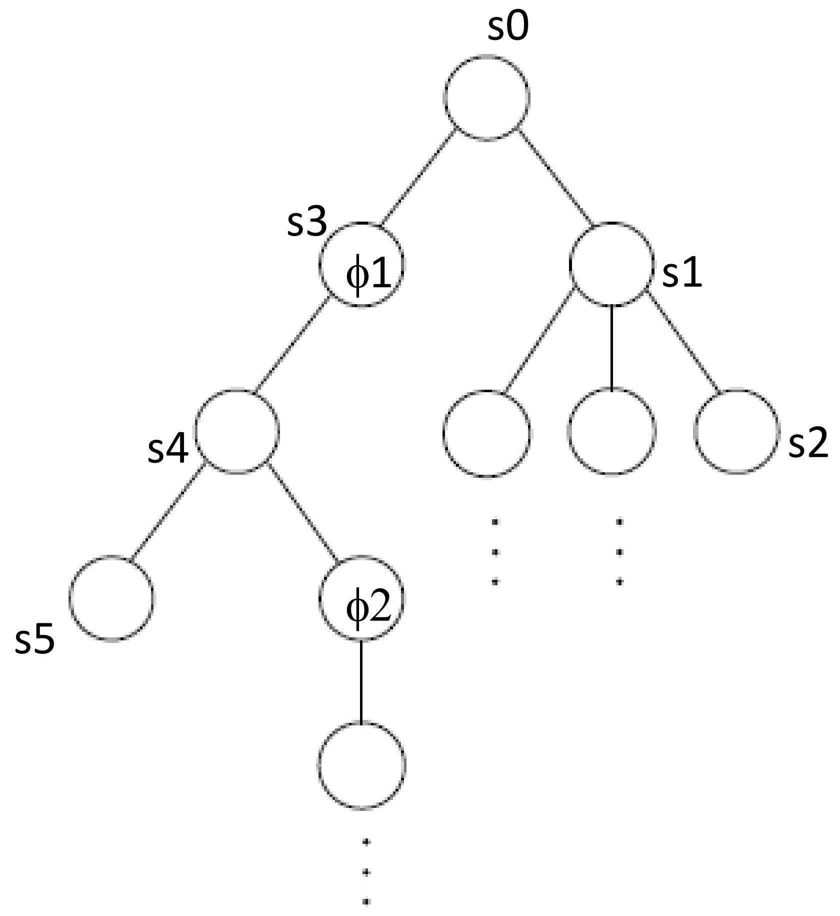
Example: counter-example facility

ϕ1: ldc *Activation*
ϕ2: ldc *HomePage*

Property 2 (P2)
For all paths: a state is reached that satisfies ϕ1 and successively a state is reached that satisfies ϕ2

S0 |= P2 FALSE

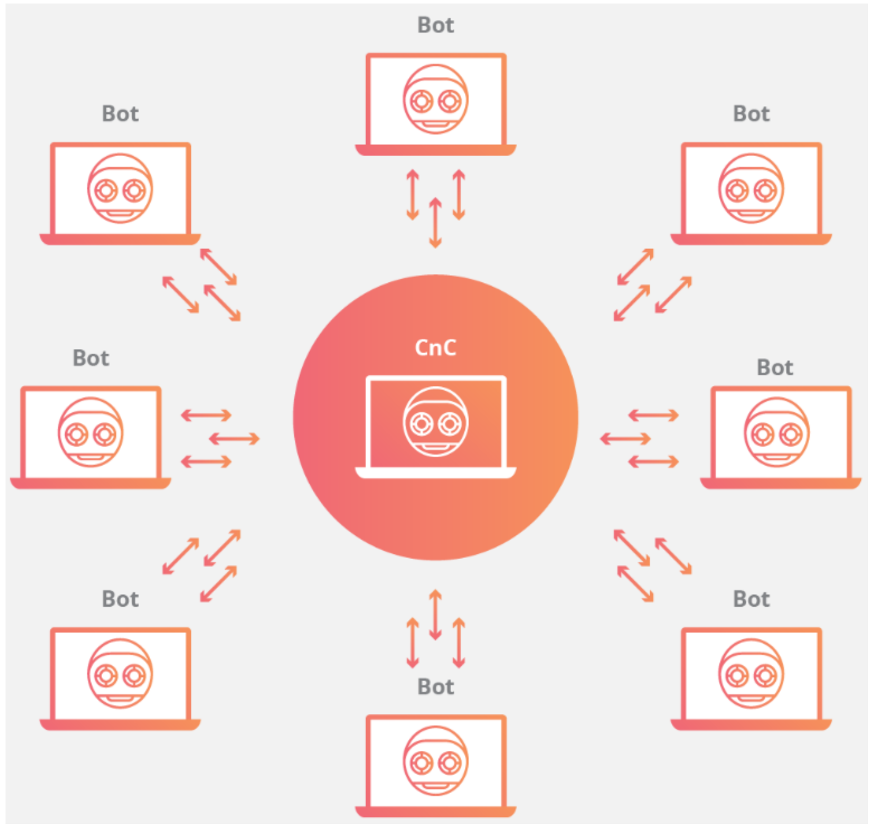
Counter-example: s0 s1 s2
Counter-example: s0 s3 s4 s5



Examples of (botnet) malware family payload

- The term mobile botnet refers to a group of compromised smartphones that are remotely controlled by botmaster susing Command & Control (C&C) channels.
- While PC-based botnets, as common platforms for many Internet attacks, they become one of the most serious threats to Internet, mobile botnets targeted for smartphones are not as popular as their counterparts for several reasons including resource issues, limited battery power, and Internet access constraints.

Botnet main schema



TigerBot Botnet Payload Detection Temporal Logic Formula

```
public a$a(a parama, Message paramMessage)
  {
    /* ... */
    this.jdField_for = paramMessage.getData().getString("packName");
    /* ... */
    this.jdField_if.jdField_char = paramMessage.getData().getString("addrType");
    this.jdField_if.jdField_case = paramMessage.getData().getBoolean("openGPS");
    /* ... */
    this.jdField_if.jdField_long = paramMessage.getData().getInt("timeOut");
    this.jdField_if.jdField_goto = paramMessage.getData().getInt("priority");
    this.jdField_if.jdField_void = paramMessage.getData().getBoolean("location_change_notify");
    /* ... */
  }
```

$$\begin{aligned}
\varphi_{TIGERBOT} &= \mu X.\langle invokegetData\rangle\, \varphi_{1_1} \vee \langle -invokegetData\rangle\, X\\
\varphi_{1_1} &= \mu X.\langle pushpackName\rangle\, \varphi_{1_2} \vee \langle -pushpackName\rangle\, X\\
\varphi_{1_2} &= \mu X.\langle invokegetString\rangle\, \varphi_{1_3} \vee \langle -invokegetString\rangle\, X\\
\varphi_{1_3} &= \mu X.\langle invokegetData\rangle\, \varphi_{1_4} \vee \langle -invokegetData\rangle\, X\\
\varphi_{1_4} &= \mu X.\langle pushprodName\rangle\, \varphi_{1_5} \vee \langle -pushprodName\rangle\, X\\
\varphi_{1_5} &= \mu X.\langle invokegetString\rangle\, \varphi_{1_6} \vee \langle -invokegetString\rangle\, X\\
\varphi_{1_6} &= \mu X.\langle invokegetData\rangle\, \varphi_{1_7} \vee \langle -invokegetData\rangle\, X\\
\varphi_{1_7} &= \mu X.\langle pushcoorType\rangle\, \varphi_{1_8} \vee \langle -pushcoorType\rangle\, X\\
\varphi_{1_8} &= \mu X.\langle invokegetString\rangle\, \varphi_{1_9} \vee \langle -invokegetString\rangle\, X\\
\varphi_{1_9} &= \ldots
\end{aligned}$$

RootSmart Botnet Payload Detection Temporal Logic Formula

```
{ /* ... */
  if ("action.host_start".equals(str)) { /* ... */ }
  if (!"action.shutdown".equals(str)) { /* ... */ }
  if (!"action.screen_off".equals(str)) { /* ... */ }
  if (!"action.install".equals(str)) { /* ... */ }
  if ("action.installed".equals(str)) { /* ... */ }
  if ("action.check_live".equals(str)) { /* ... */ }
  if ("action.download_shells".equals(str)) { /* ... */ }
  if ("action.exploid".equals(str)) { /* ... */ }
  if ("action.first_commit_localinfo".equals(str)) { /* ... */ }
  if ("action.second_commit_localinfo".equals(str)) { /* ... */ }
  if ("action.load_taskinfo".equals(str)) { /* ... */ }
  if ("action.download_apk".equals(str)) { /* ... */ }
    /* ... */
}
```

$$
\begin{aligned}
\varphi_{ROOTSMART} &= \mu X.\langle pushactionhoststart\rangle\, \varphi_{2_1} \vee \langle -pushactionhoststart\rangle X\\
\varphi_{2_1} &= \mu X.\langle invokeequals\rangle\, \varphi_{2_2} \vee \langle -invokeequals\rangle X\\
\varphi_{2_2} &= \mu X.\langle ifeq\rangle\, \varphi_{2_3} \vee \langle -ifeq\rangle X\\
\varphi_{2_3} &= \mu X.\langle pushactionboot\rangle\, \varphi_{2_4} \vee \langle -pushactionboot\rangle X\\
\varphi_{2_4} &= \mu X.\langle invokeequals\rangle\, \varphi_{2_5} \vee \langle -invokeequals\rangle X\\
\varphi_{2_5} &= \mu X.\langle ifeq\rangle\, \varphi_{2_6} \vee \langle -ifeq\rangle X\\
\varphi_{2_6} &= \mu X.\langle pushactionshutdown\rangle\, \varphi_{2_7} \vee \langle -pushactionshutdown\rangle X\\
\varphi_{2_7} &= \mu X.\langle invokeequals\rangle\, \varphi_{2_8} \vee \langle -invokeequals\rangle X\\
\varphi_{2_8} &= \mu X.\langle ifeq\rangle\, \varphi_{2_9} \vee \langle -ifeq\rangle X\\
\varphi_{2_9} &= \ldots
\end{aligned}
$$

Conclusions

- On overview about malware taxonomy and classification
- We generated a X86 payload
- We discussed about Android security
- We use (real-world) tools for Android malware analysis for finding interesting pattern
- We explained how formal methods can be useful for effective malware family detection
- In the future detection will be more and more difficult
  - because malware is increasing its obfuscation capabilities...

References

- Gerardo Canfora, Fabio Martinelli, Francesco Mercaldo, Vittoria Nardone, Antonella Santone, Corrado Aaron Visaggio: ***LEILA: Formal Tool for Identifying Mobile Malicious Behaviour***. IEEE Trans. Software Eng. 45(12): 1230-1252 (2019)

- Cinzia Bernardeschi, Francesco Mercaldo, Vittoria Nardone, Antonella Santone: ***Exploiting Model Checking for Mobile Botnet Detection***. KES 2019: 963-972

- Aniello Cimitile, Francesco Mercaldo, Vittoria Nardone, Antonella Santone, Corrado Aaron Visaggio: ***Talos: no more ransomware victims with formal methods***. Int. J. Inf. Sec. 17(6): 719-738 (2018)

- Francesco Mercaldo, Vittoria Nardone, Antonella Santone, Corrado Aaron Visaggio: ***Download malware? no, thanks: how formal methods can block update attacks***. FormaliSE@ICSE 2016: 22-28