



Malware Behavioral Detection by Attribute-Automata using Abstraction from Platform and Language

JACOB Grégoire^{1/2}, DEBAR Hervé^{1/2}, FILIOL Eric²

¹ *Orange Labs / France Télécom R&D,
Security and Trusted Transactions (MAPS/STT).*

² *ESIEA, Cryptology & Virology Lab.*

12th RAID Symposium
September 2009

1. Outline

Context

- Interest of behavioral detection against unknown malware
*Theoretically detects, if not innovative malware,
at least variants reusing known techniques*
- In AV products, behavioral detectors still rely on too specific characteristics
Escape through simple functional modifications (variants multiplication)

Problematics

- Can we describe malicious behaviors generically ?
- Can we address the semantic gap between the model and data collection ?
- Can we detect accurately these descriptions in a reasonable time ?

1. Outline

Increasing expressiveness of behavioral models

- 1995 – Simple Finite State Automata [B. L. Charlier et al.]
 - Alternative sequences of operations
- 2005 – Information flow analysis [J. Newsome et al., S. Bhatkar et al.]
 - Operations involving misappropriate data flow
- 2007 – Graphs with data dependencies [M. Christodorescu et al., L. Martignoni et al., J. Morales et al.]
 - Sequences of operations with data dependencies

Summary

1 ■ Outline

2 ■ Behavioral descriptions based on attribute-grammars

- Abstract Malicious Behavior Language
- Describing duplication

3 ■ Detection by attribute-automata

- Layered architecture
- Abstraction layer for translation
- Detection layer by attribute-automata
- Prototyping

4 ■ Coverage and performance evaluation

- Detection and errors rates
- Performance

5 ■ Considerations and perspectives

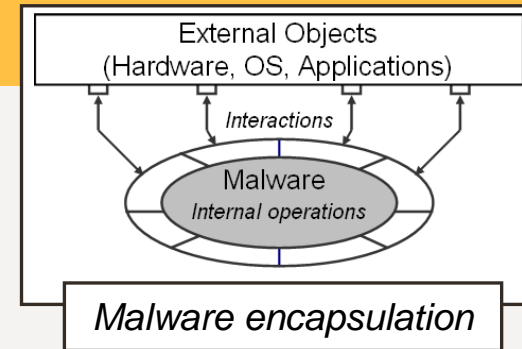
2

Behavioral Descriptions based on Attribute-Grammars

2.1 Abstract Malicious Behavior Language

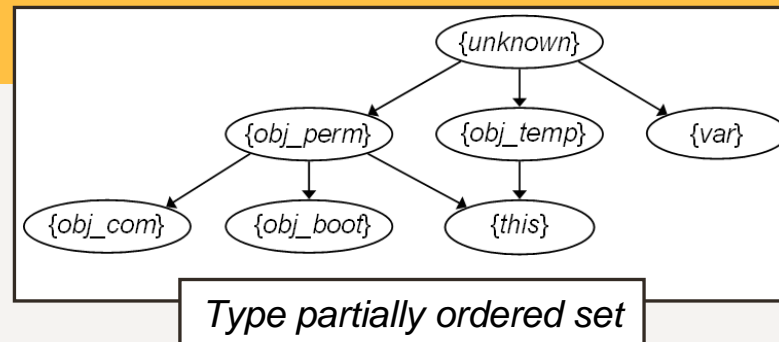
Object-oriented principles

- Internal operations (Turing complete)
- Interactions to interface with external objects
- Grammar: syntax and operational semantics for operations and interactions



Above semantic rules

- Object binding using identifiers
 - Constraints on the data-flow
- Object typing
 - Reveals the purpose of objects in the malware lifecycle e.g.



Purpose	Type
Persistence	Permanent objects
Propagation	Communicating objects
Residency	Booting objects

2.2 Describing Duplication

Duplication

- Duplication principle:
Copying data from the self-reference towards a permanent object
- Syntactic productions convey different technical solutions:
 - Single block read/write
 - Interleaved read/write
 - Direct copy
 - Possible permutations
- Propagation differs in typing:
Communicating object as target

(i)	<code><Duplication></code>	::=	<code><Create><Open><Read><Write></code>	
			<code><Open><Create><Read><Write></code>	
			<code><Open><Read><Create><Write></code>	
{	<code><Duplication>.srcId</code>	=	<code><Open>.objId</code>	
	<code><Duplication>.srcType</code>	=	<code>this</code>	Object typing
	<code><Duplication>.targId</code>	=	<code><Create>.objId</code>	
	<code><Duplication>.targType</code>	=	<code>obj_perm</code>	
	<code><Create>.objType</code>	=	<code><Duplication>.targType</code>	
	<code><Open>.objType</code>	=	<code><Duplication>.srcType</code>	
	<code><Read>.objId</code>	=	<code><Duplication>.srcId</code>	Object binding
	<code><Read>.objType</code>	=	<code><Duplication>.srcType</code>	
	<code><Write>.objId</code>	=	<code><Duplication>.targId</code>	
	<code><Write>.objType</code>	=	<code><Duplication>.targType</code>	
	<code><Write>.varId</code>	=	<code><Read>.varId</code>	
}				
			<code><Open><Create><InterleavedRW></code>	
			<code><Create><Open><InterleavedRW></code>	
{	<code><InterleavedRW>.obj1Id</code>	=	<code><Duplication>.srcId</code>	
	<code><InterleavedRW>.obj1Type</code>	=	<code><Duplication>.srcType</code>	
	<code><InterleavedRW>.obj2Id</code>	=	<code><Duplication>.targId</code>	
	<code><InterleavedRW>.obj2Type</code>	=	<code><Duplication>.targType</code>	
}				
			<code><DirectCopy></code>	
{	<code><Duplication>.srcId</code>	=	<code><DirectCopy>.obj1Id</code>	
	<code><Duplication>.srcType</code>	=	<code>this</code>	
	<code><Duplication>.targId</code>	=	<code><DirectCopy>.obj2Id</code>	
	<code><Duplication>.targType</code>	=	<code>obj_perm</code>	
	<code><DirectCopy>.obj1Type</code>	=	<code><Duplication>.srcType</code>	
	<code><DirectCopy>.obj2Type</code>	=	<code><Duplication>.targType</code>	
}				

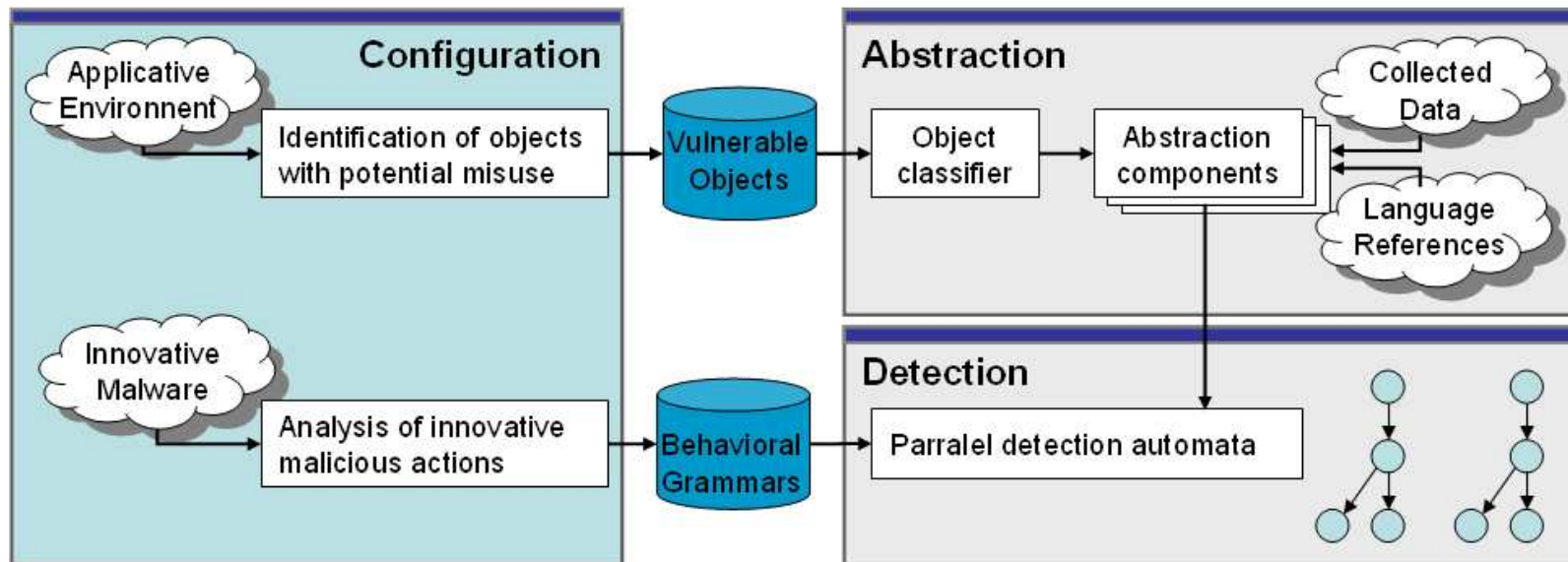
3

Detection by Attribute-Automata

3.1 Layered Architecture


Global architecture in separate layers

- Collection mechanisms: *recovers execution traces*
- Abstraction layer: *translates collected traces into the behavioral model*
- Detection by parallel attribute-automata: *parses behavior descriptions*
- Configuration process: *new objects, languages, or behaviors*



3.2 Abstraction Layer for Translation

Translating operations and interactions

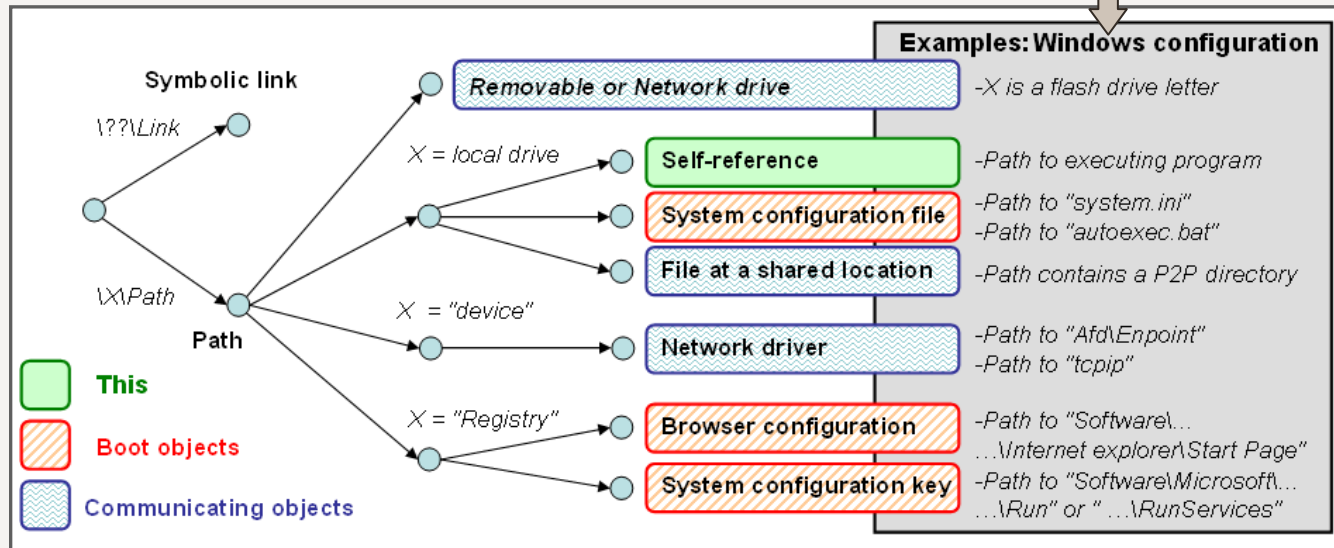
- Translation is specific to a given language
- Translation by mapping for arithmetic and control operations
- Translation by mapping from API calls over interactions 

Interaction Class	Object Nature	Windows Native API	VBScript API
Open	File	NtOpenFile (ptr FileHandle, ..., str FilePath, ...) NtCreateSection (ptr SectionHandle, ..., ptr FileHandle)	FileSystemObject. GetFile (str FilePath) FileSystemObject. GetFolder (str FilePath) FileSystemObject. OpenTextFile (str FilePath) FileSystemObject. GetDrive (str DrivePath) FileSystemObject. Drives.Item (int DriveNumber)
	Registry	NtOpenKey (ptr KeyHandle, ..., str KeyName, ...) NtEnumerateKey (ptr KeyHandle, ...)	
	Network	NtOpenFile (ptr DeviceHandle, ..., str NetworkDevicePath, ...)	
Write	File	NtWriteFile (ptr FileHandle, ..., ptr Buffer, ...) NtWriteFileGather (ptr FileHandle, ..., ptr SegmentArray, ...)	FileObject. Write (str Value) FileObject. WriteLine (str Value) FileObject. Copy (str FilePath) FileObject. Move (str FilePath) FileSystemObject. CopyFile (str FilePath, str FilePath) FileSystemObject. MoveFile (str FilePath, str FilePath)
	Registry	NtSetValueKey (ptr KeyHandle, str Value, ..., str Buffer, ...)	ShellObject. RegWrite (str KeyName, str Value)
	Network	NtDeviceIoControlFile (ptr DeviceHandle, ..., SendControl, ptr Buffer, ...)	
	Mail		MailObject. TextBody (str Content) MailObject. Body (str Content) MailObject. AddAttachment (str FilePath) MailObject. AttachFile.Add (str FilePath) MailObject. Attachments.Add (str FilePath)

3.2 Abstraction Layer for Translation

Translating external objects

- Translation affects to objects a unique identifier and a type
- Specific to a platform and its applicative configuration
- Deployed by decision trees depending on the object representation:
constants, addresses and handles, character strings

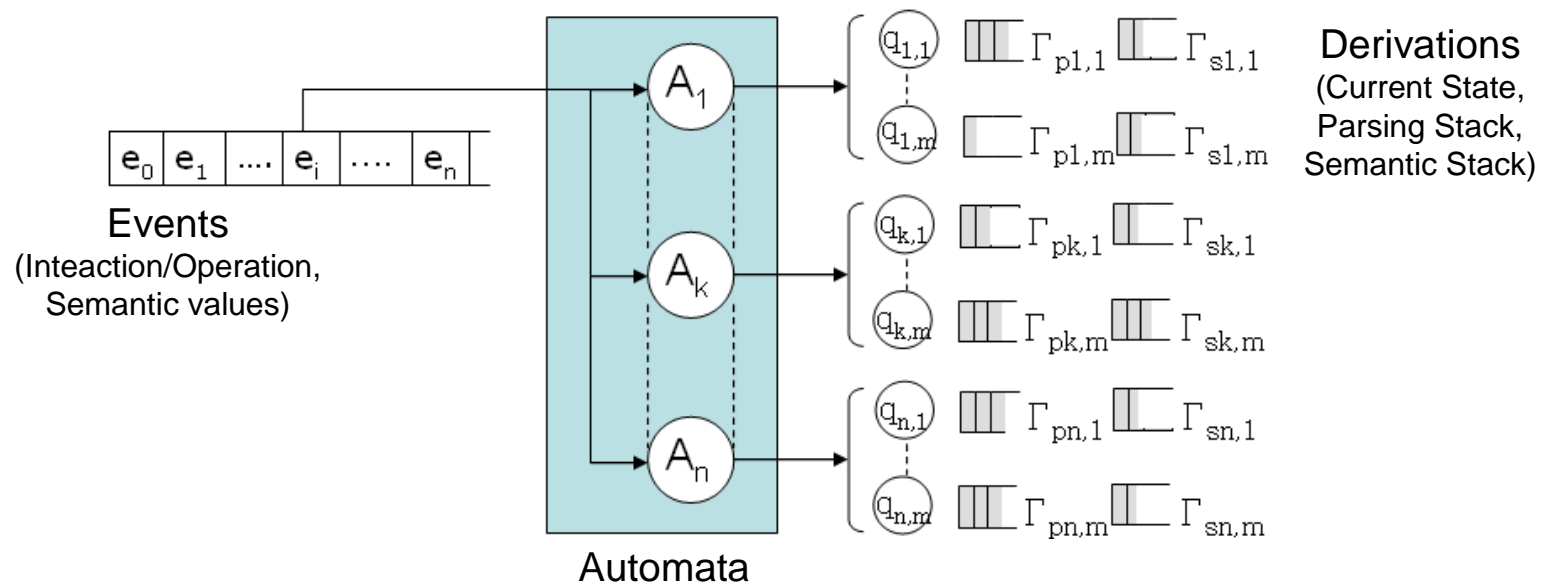


- Tree generation by identification of vulnerable objects at three levels:
hardware, operating system, applications (connected and widely deployed)

3.3 Detection Layer by Attribute-Automata

Algorithm properties

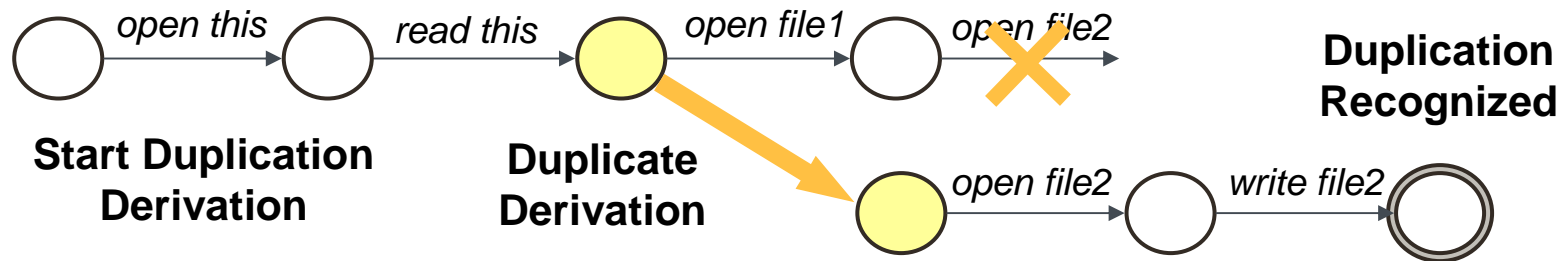
- Translated events in input
- Each event feeds the parallel automata for progression in the derivations
- Each automaton manage several derivations:
parallel derivations corresponds to different behavior instances



3.3 Detection Layer by Attribute-Automata

Algorithm properties

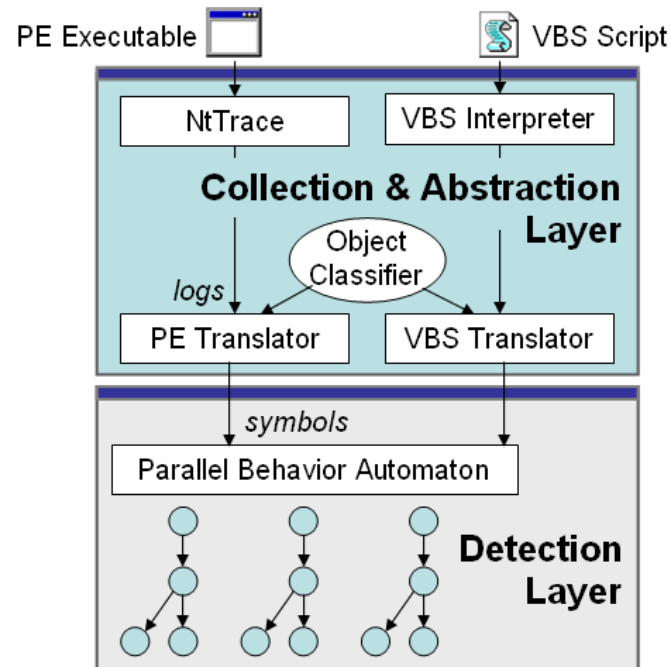
- Semantic routines check prerequisites and evaluate consequences:
match collected semantic values with computed ones
or computes new values from existing ones
- Irrelevant events are discarded
- Potentially ambiguous events duplicate derivations:
Ambiguous = related to the behavior but making derivation fail



3.4 Prototyping

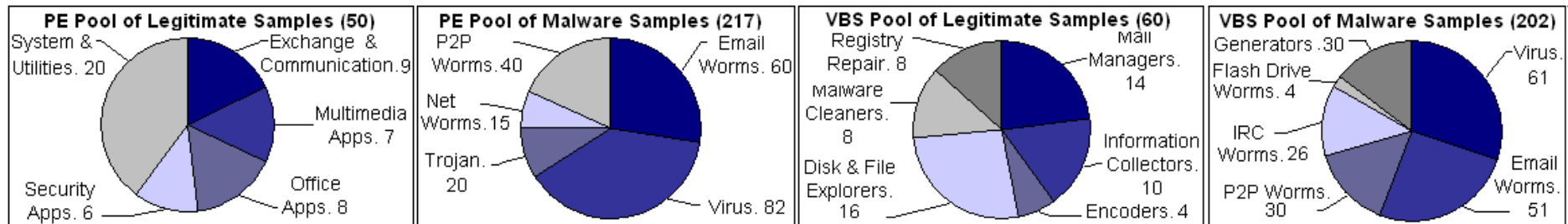
Prototype global architecture

- Two abstraction components for PE traces and VBS Scripts:
log analysis for PE dynamic traces and path exploration for VBS scripts
- Four detection automata:
duplication, propagation, residency and overinfection tests



4

Coverage and Performance Evaluation



4.1 Detection and Error Rates

Detection rates by behavior

Behaviors	EmW	P2PW	V	NtW	Trj	Global
Duplication	41(68,33%)	31(77,5%)	15(18,29%)	8(53,33%)	6(30%)	46,54%
By single read/write	41(68,33%)	30(75%)	14(17,07%)	8(53,33%)	6(30%)	45,63%
By interleaved read/write	0(15%)	3(7,5%)	3(3,66%)	3(0,2%)	0(0%)	8,29%
Propagation FN	4(6,67%)	19(47,5%)	3(3,66%)	1(6,67%)	0(0%)	12,44%
By single read/write	4(6,67%)	19(47,5%)	3(3,66%)	1(6,67%)	0(0%)	12,44%
Residency	36(60%)	22(55%)	5(60,98%)	6(40%)	9(45%)	35,94%
Overinfection test	0(0%)	0(0%)	0(0%)	0(0%)	0(0%)	0,00%
Global detection	43(71,67%)	33(82,50%)	16(19,51%)	8(53,33%)	11(55,00%)	51,15%

PE Detection Rates
 EmW = Email-Worms,
 P2PW = P2P-Worms,
 NtW = Network Worms,
 V = Virii, Trj = Trojans

VBS Detection Rates EmW = Email-Worms,
 FdW = Flashdrive-Worms, IrcW = Irc-Worms, V = Virii,
 P2PW = P2P-Worms, Gen = Malware Generators

Behaviors	EmW	FdW	IrcW	P2PW	V	Gen	Global
Duplication	43(84,31%)	4(100%)	20(76,96%)	22(73,33%)	44(72,13%)	30(100%)	80,70%
By direct copy	41(80,39%)	4(100%)	20(76,96%)	22(73,33%)	25(40,98%)	30(100%)	70,30%
By single read/write	8(15,69%)	0(0%)	4(15,38%)	3(10%)	21(34,43%)	0(0%)	17,82%
By interleaved read/write	1(1,96%)	0(0%)	0(0%)	0(0%)	8(13,11%)	0(0%)	4,46%
Propagation	33(64,71%)	3(75%)	5(19,23%)	25(83,33%)	5(8,20%)	30(100%)	49,99%
By direct copy	33(64,71%)	3(75%)	4(15,38%)	25(83,33%)	3(4,92%)	30(100%)	48,52%
By single read/write	3(5,88%)	0(0%)	2(7,69%)	1(3,33%)	2(3,28%)	0(0%)	3,96%
Residency	32(62,75%)	4(100%)	20(76,92%)	18(60,00%)	20(32,79%)	30(100%)	61,39%
Overinfection test	4(7,84%)	1(25%)	1(3,85%)	0(0%)	0(0%)	0(0%)	2,97%
By conditional 1	4(7,84%)	1(25%)	1(3,85%)	0(0%)	0(0%)	0(0%)	2,97%
By inverse conditional 2	0(0%)	0(0%)	0(0%)	0(0%)	0(0%)	0(0%)	0,00%
Global detection	46(90,20%)	4(100%)	25(96,15%)	27(90,00%)	50(81,97%)	30(100%)	90,09%

TP

FN

TP

4.1 Detection and Error Rates

False negatives

- Limitations in the dynamic collection mechanisms (PE Traces Analyzer)
 - Simulation software configuration: *64% of missed Virii did not execute properly*
 - Simulation network configuration: *75% of Email-Worms did not show SMTP activity*
 - Collection level impacting the data-flow: *10% of Virii and Email-Worms missed because of intermediate operations in memory (mutation, base64 encoding)*
- Limitations in the static collection mechanisms (VBS Script Analyzer)
 - Body ciphering: *only string ciphering supported yet*
 - Cohabitation with other languages: *failure of the syntactic analysis*
- Irrelevances in the behavioral model
 - Too much specific descriptions: *only 2% of Overinfection tests detected*

False positives for legitimate samples

- A single false positive for residency in more than a hundred samples
 - No real false-positive: *malware cleaner restarting the browser start page*

4.2 Performance

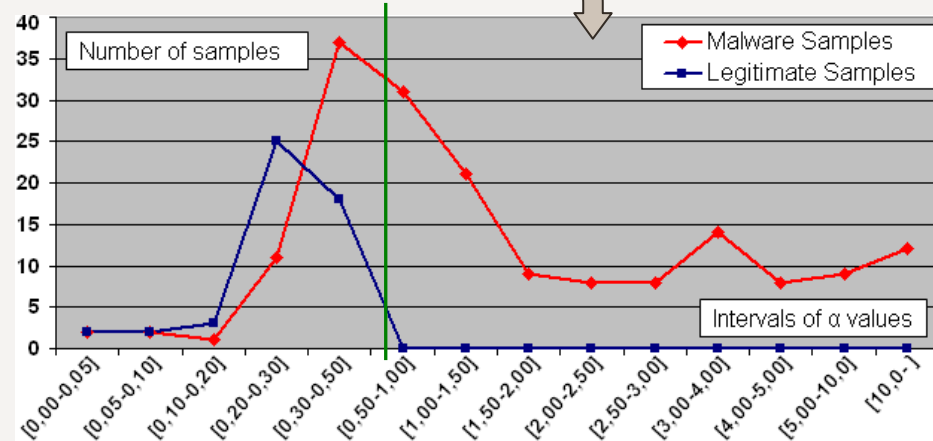
Material performance (Dual Core 2,6GHz)

- PE Traces Analyzer: 0,340s/log
- VB Script Analyzer: 0,016s/log
- Detection Automata: 0,440s/log (PE) 0,001s/log (VBS)

Detection complexity

- Important theoretical complexity in worst case scenario
- Reasonable operational complexity in function of the ambiguity ratio α
- An important α is already a sign of malicious activity

Complexity	Value
Worst Case	$\vartheta(k(2^n - 1))$
Best Case	$\vartheta(kn)$
Operational	$\vartheta(k\alpha(\frac{n^2+n}{2}))$



5

Considerations and Perspectives

5. Considerations and perspectives

Contributions

- Generic, synthetic and human understandable behavioral signatures
- Proofs of concept for the detection automata and two abstraction components analyzing PE traces and VBS scripts
- Experimentations showing promising detection rates and reasonable performances

Perspectives

- Increase the detection coverage by using sophisticated collection tools: *tainting tools to avoid breakdowns in the data flow*
- Profiling malware categories according to their behaviors

Thank you for your attention,



Any questions?