Semantic Trace-based Malware Variants Detection

Khalid Alzarooni

CREST - DCS - UCL

April 6, 2011

Outline

Overview

Malware Variants

- Speed of evolution of malware partly driven by automatic generation of program variants
- Semantic equivalence tables used in malware, e.g. polymorphic and metamorphic malware
- These alter "local behaviour" of programs but larger scale behaviour is unchanged

Malware Problem

Anoirel S. Issa
Symantec, UK (EICAR 2009)

> *"Poly or metamorphic engines have some essential components that help them build highly obfuscated code. A single engine is able to produce unique variants that can reach millions."*

Malware evolution: $M_0 \rightarrow M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow \ldots$
Syntactic view: $code_0 \not\approx code_1 \not\approx code_2 \not\approx code_3 \not\approx \ldots$

Some Code Obfuscation Schemes

[Beaucamps, 2007, Ször, 2005]

Label	Category	Obfuscation
gi	Garbage insertion	$\{\} \rightarrow \{C\}$
op	Opaque predicate	$\{\} \rightarrow \{P^{T/F}\}$
ec	Equivalent command	$\{op\} \rightarrow \{\bar{o}p\}$
rr	Register renaming	$\{Rx\} \rightarrow \{Ry\}$
cs	Command split	$\{C\} \rightarrow \{C_x, C_y\}$
cm	Command merging	$\{C_x, C_y\} \rightarrow \{C_{xy}\}$
cr	Command reorder	$\{(C_x, C_y)\} \rightarrow \{(C_y, C_x)\}$
..	...	...

Example: a program P and its semantically equivalent variant P'

P :

a	`R0:=n`
b	`R1:=m`
c	`R2:=R1`
d	`R3:=R2+R0`
e	`R4:=R1+k`
f	`R5:=1`

$\longrightarrow$

P' :

a'	`R0:=n`
cr_1	`JMP` rr_1
gi_1	`R22:=R22+1`
op_1	P^T `JMP` cm
rr_1	`R11:=m`
gi_2	`R22:=R22+1`
cr_2	`JMP` op_1
cm	`R3:=R11+R0`
e'_1	`R4:=k`
e'_2	`R4:=R4+R11`
rr_2	`R15:=1`

Malware Problem

- To detect variants of a known malware
- Given two arbitrary programs is it possible to tell whether they are semantically equivalent?
- It is undecidable: not possible to devise an algorithm to produce "yes" or "no" detection answer [Cohen, 1987]

$$P' \stackrel{?}{\approx} P$$

Semantic trace-based

Program

↓

Program approximation

↓

Trace collection

↓

Semantic analysis

↓

Detection of semantic signatures

Test scenarios

Results:

- Tested samples: Bho, Binom, Mobler, Telf, ...
- Most malware successfully matched, with $k \geq 60\%$
- No false positives, similarity $\leq 20\%$ (10 benign executables)
- 100% malware variants classification
- sig-w-slice: accuracy 30% and speed 26% in detection phase
- sig-wo-slice: 5 : 7 faster in sig. generation phase

Trace-based approach

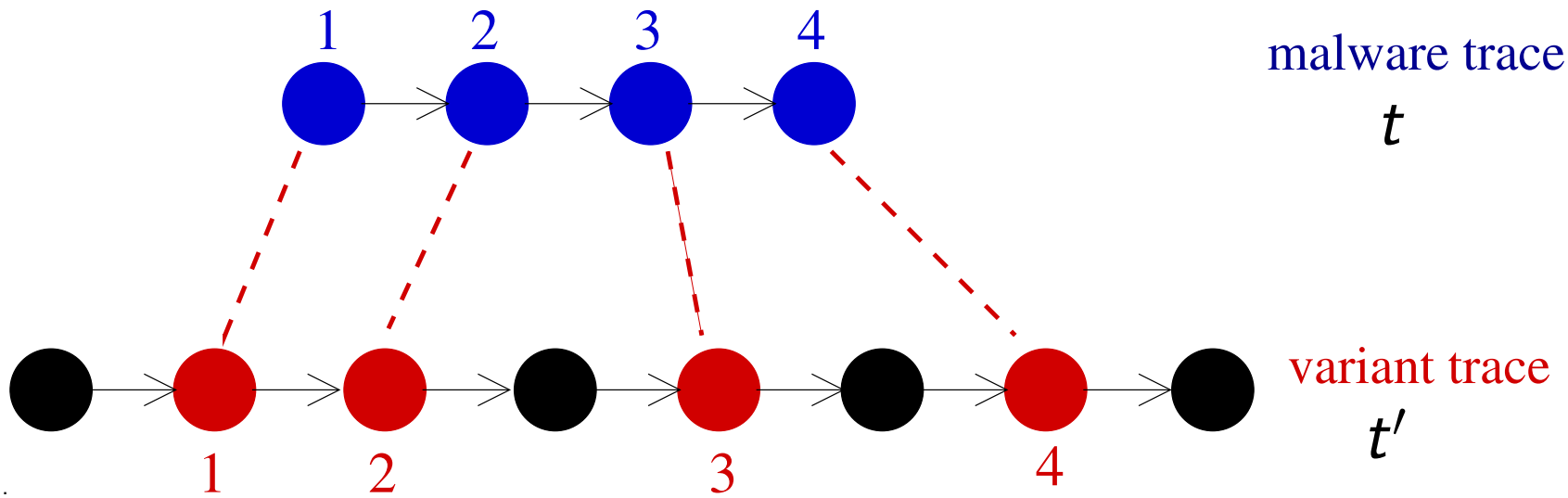
Semantic trace-based

- Design a detector that can tell when two programs are *approximately* equivalent, which might often be good enough
- Approximate semantic equivalence is decidable
- Approximate a program's semantics $[\![P]\!]$
  - CFG abstract traces (program paths) & test inputs
  - concrete & semantic traces

Malware evolution: $M_0 \rightarrow M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow \ldots$
Syntactic view: $code_0 \not\approx code_1 \not\approx code_2 \not\approx code_3 \not\approx \ldots$
Semantic view: $[\![M_0]\!] \approx [\![M_1]\!] \approx [\![M_2]\!] \approx [\![M_3]\!] \approx \ldots$

Semantic trace-based

- M_1 is a variant of M_0 if $[\![M_0]\!]$ is sub-sequence of $[\![M_1]\!]$



$$\forall t \in [\![M_0]\!],\ \exists t' \in [\![M_1]\!] :\ t \prec t'$$

Semantic trace-based

Two phases:

1. Signature generation
2. Detection

Signature generation phase

executable M

$\downarrow$ (disassembler & translator)

abstract code (AAPL)

$\downarrow$ (test data generator)

abstract trace and a test input $\mathtt{x}$

$\downarrow$ (semantic simulator)

a concrete trace

$\downarrow$ (trace slicer)

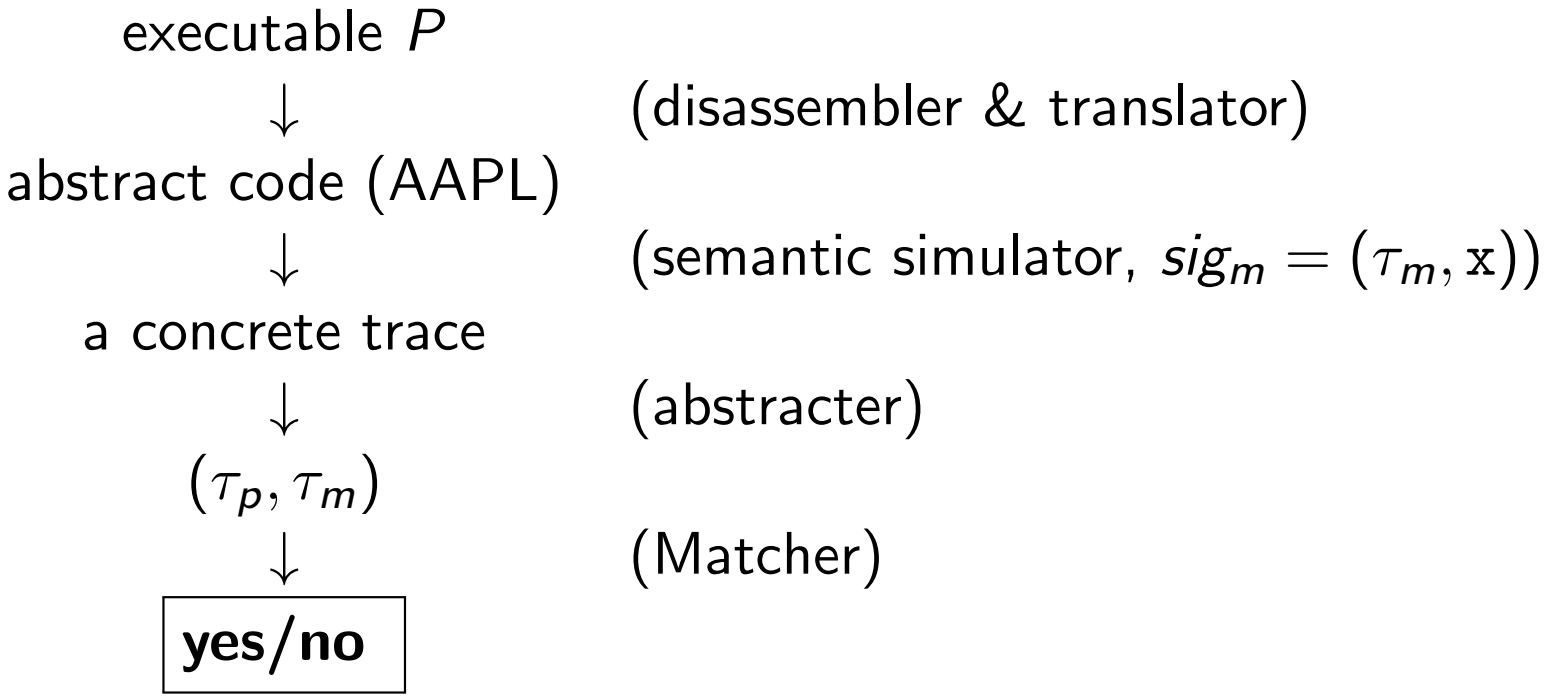
trace slices

$\downarrow$ (abstracter)

semantic traces τ_m

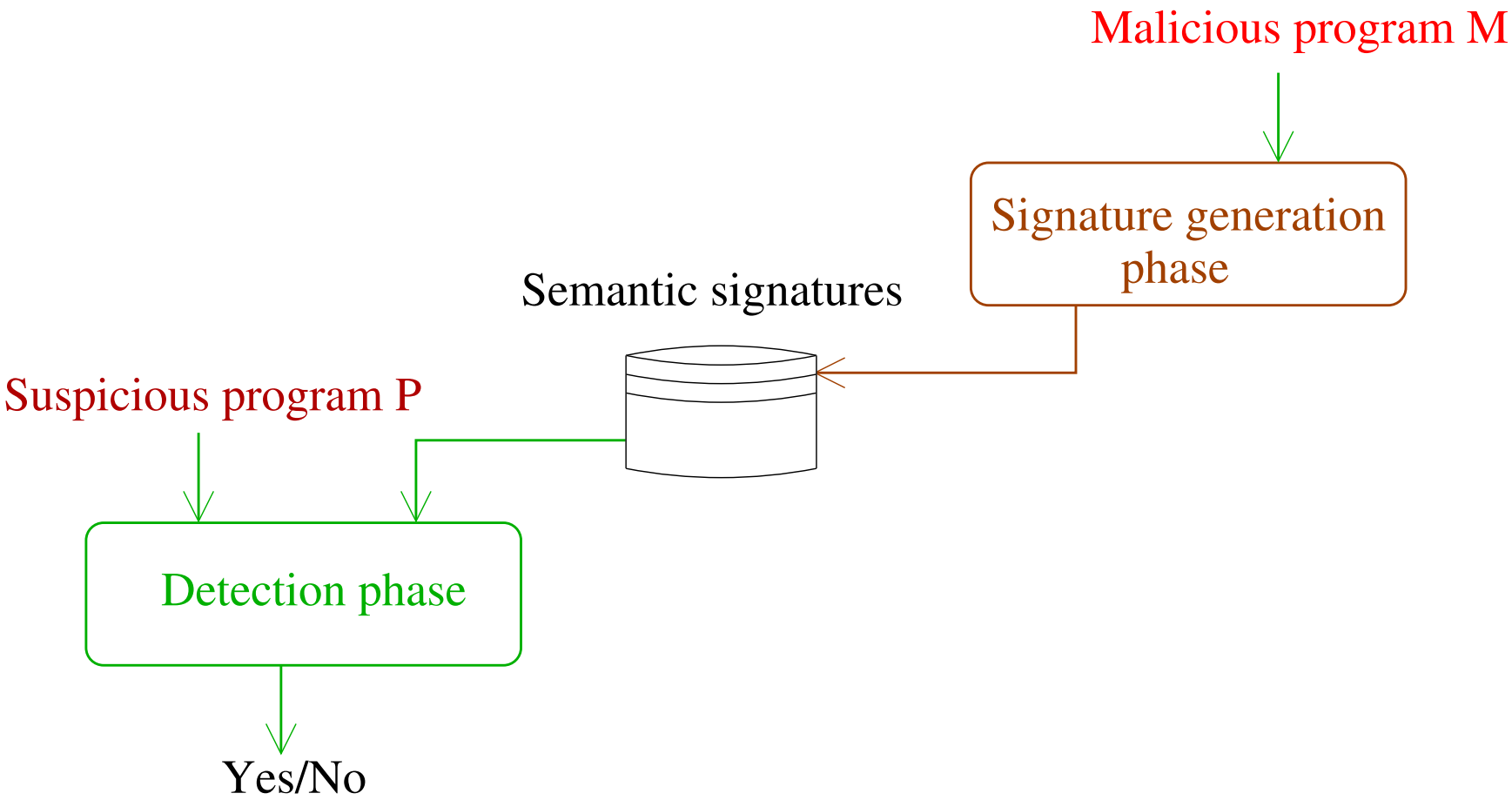
semantic signature $= (\tau_m, \mathtt{x})$

Detection phase



Experiments

Detector prototype



Test scenarios

We tested:

- Robustness against real in-the-wild variants
- Effectiveness of trace slicing in the signatures
- Fig. gen.& detection phases: sig-wo-slice vs. sig-w-slice
- False positives
- Classification of malware samples

Test scenarios

Results:

- Tested samples: Bho, Binom, Mobler, Telf, ...
- Most malware successfully matched, with $k \geq 60\%$
- sig-w-slice: accuracy 30% and speed 26% in detection phase
- sig-wo-slice: 5 : 7 faster in sig. generation phase
- No false positives, similarity $\leq 20\%$ (10 benign executables)
- 100% malware variants classification

Prototype limitation

Technical shortcomes:

- Limited to viruses and worms
- Does not work for *dynamic* packed code and code with *anti-disassembly* techniques and
- Relay on tools to manually unpack (encrypted) and disassemble files

Thank you very much !

[0]Image: Salvatore Vuono / FreeDigitalPhotos.net

References

Alzarouni, K., Clark, D., and Tratt, L. (2010).

Semantic malware detection.

Technical Report TR-10-03, Department of Computer Science, King's College London.

Beaucamps, P. (2007).

Advanced metamorphic techniques in computer viruses.

In *Proceedings of the International Conference on Computer, Electrical, and Systems Science, and Engineering - CESSE'07*.

Cohen, F. (1987).

Computer viruses: theory and experiments.

Comput. Secur., 6(1):22–35.

Ször, P. (2005).

The Art of Computer Virus Research and Defense.

Addison-Wesley, Reading, Mass.

Detector components

Trace Semantics

- Trace semantics of a program is the set of all traces T that the program can produce
- A trace $t \in T$ is a sequence of pairs of execution context $\mathcal{X}$ and program syntax $\mathcal{C}$
- Execution context: memory (locations) and environment (variables) values $\mathcal{X} = \mathcal{E} \times \mathcal{M}$
- Program syntax: source code (commands)

$$\rho \in \mathcal{E} = \mathbf{R} \to \mathbb{Z}_\perp \qquad \text{(environments)}$$
$$m \in \mathcal{M} = \mathbb{Z} \to \mathbb{Z}_\perp \cup \mathbf{C} \qquad \text{(memory)}$$
$$\xi \in \mathcal{X} = \mathcal{E} \times \mathcal{M} \qquad \text{(execution contexts)}$$
$$\mathcal{S} = \mathbf{C} \times \mathcal{X} \qquad \text{(program states)}$$

Trace Semantics

- Signatures refer to exact program state
- Semantic signatures refer to values at particular memory locations and in registers that are observed to be constant across variants from the same malware family
- Detection: environment-memory traces of M that are contained (subtraces) of environment-memory traces of M'

Semantic Simulator

Not "live" testing

Evaluate abstract trace and collect concrete traces

Semantics of Actions:

$$\hat{\mathbf{A}} : \mathbf{A} \times \mathcal{X} \to \mathcal{X}$$

$$\hat{\mathbf{A}}[\![R := E]\!]\xi = (\rho', m) \qquad \text{where } \xi = (\rho, m) \text{ and } \rho' = \rho(R \mapsto \hat{\mathbf{E}}[\![E]\!]\xi)$$

$$\hat{\mathbf{A}}[\![*R := E]\!]\xi = (\rho, m') \qquad \text{where } \xi = (\rho, m) \text{ and } m' = m(\rho(R) \mapsto \hat{\mathbf{E}}[\![E]\!]\xi)$$

$$\hat{\mathbf{A}}[\![\texttt{JMP E}]\!]\xi = (\rho', m) \qquad \text{where } \xi = (\rho, m) \text{ and } \rho' = \rho(PC \mapsto \hat{\mathbf{E}}[\![E]\!]\xi)$$

$$\hat{\mathbf{A}}[\![\texttt{RTN}]\!]\xi = (\rho', m) \qquad \text{where } \xi = (\rho, m) \text{ and } \rho' = \rho(PC \mapsto m(\rho(SP)), SP \mapsto SP + 1)$$

$$\hat{\mathbf{A}}[\![\texttt{PUSH E}]\!]\xi = (\rho', m') \qquad \text{where } \xi = (\rho, m) \text{ and } \rho' = \rho(SP \mapsto SP - 1) \text{ and } m' = m(\rho(SP - 1) \mapsto \hat{\mathbf{E}}[\![E]\!]\xi)$$

Semantic Simulator

Not "live" testing
Evaluate abstract trace and collect concrete traces
Semantics of Commands:

$\hat{\mathbf{C}} : S \to \Sigma(S)$ (determines transition relation between states)

$\hat{\mathbf{C}}[\![C_A]\!]\xi = (\xi', C')$ where $\xi = (\rho, m)$, $\xi' = \hat{\mathbf{A}}[\![A]\!]\xi$ and

$$C' = \begin{cases} m(\rho(PC)) & \text{if } A := \texttt{JMP} \cup \texttt{CALL} \cup \texttt{RTN} \\ m(\rho(PC+1)) & \text{otherwise} \end{cases}$$

$\hat{\mathbf{C}}[\![C_B]\!]\xi = (\xi', C')$ where $\xi = (\rho, m)$, and

$$(\xi', C') = \begin{cases} \xi' = (\rho', m), \rho' = \rho(PC \mapsto \hat{\mathbf{E}}[\![E]\!]\xi), C' = m(\rho(\hat{\mathbf{E}}[\![E]\!]\xi)) & \text{if } \hat{\mathbf{B}}[\![B]\!]\xi = \text{true} \\ \xi' = \xi, C' = m(\rho(PC+1)) & \text{otherwise} \end{cases}$$

TSAlgo – Trace slicing

- $P \xrightarrow{slice} P'$ (semantically invariant subprogram wrt a criterion)
- $t \xrightarrow{slice} t'$ (semantically invariant subtrace wrt *tsc*)
- Trace slicing criterion *tsc*: recent definition points of variables in t
- A conjecture: useful in the detection step for more accurate and efficient results.
- Effect is to shorten the trace and thus the signature

Signature matching

$sig = (\tau_m, \mathtt{x})$ of M and τ_p of P:

$$\mathcal{MD}(sig, P) = \begin{cases} \text{yes} & \text{if } \tau_m \text{ is contained in } \tau_p \\ \text{no} & \text{otherwise} \end{cases}$$

Our assumption: some core semantic values in the two variants that would match with a high degree of similarity, indicating the likelihood of them being behaviourally the same.

Signature matching

- we look for corresponding semantic traces of τ_m in τ_p,
- a fuzzy matching to determine whether τ_p corresponds semantically to τ_m

$$semantics\ similarity\ measure = no.\ of\ mappings/|\tau_m|$$

- We consider τ_m is contained in τ_p if the similarity measure is above a certain similarity threshold k,

$$k \leq similarity\ measure \leq 100$$

k: a large percentage of (desired) mappings