Chasing infections by unveiling program dependencies

Mila Dalla Preda[1] and Isabella Mastroeni[2]

[1] Dipartimento di Informatica - Univ. di Bologna, Italy
dallapre@cs.unibo.it
[2] Dipartimento di Informatica - Univ. di Verona, Italy
isabella.mastroeni@univr.it

Abstract. Metamorphic malware continuously modify their code, while preserving their functionality, in order to foil misuse detection. The key for defeating metamorphism relies in a semantic characterization of the embedding of the malware into the target program. Indeed, a behavioral model of program infection that does not relay on syntactic program features should be able to defeat metamorphism. Moreover, a general model of infection should be able to express dependences and interactions between the malicious code and the target program. ANI is a general theory for the analysis of dependences of data in a program. We propose an high order theory for ANI, later called HOANI, that allows to study program dependencies. Our idea is then to formalize and study the malware detection problem in terms of HOANI.

1 Introduction

One of the major challenge in computer security is the detection and neutralization of metamorphic malware. A metamorphic malware is a malicious program equipped with a metamorphic engine that takes the malware, or parts of it, as input and morphs it at run-time to a syntactically different but semantically equivalent variant, in order to foil traditional misuse malware detectors. Misuse detectors are syntactic in nature as they identify malware infection by comparing the byte sequence comprising the body of the malware against a signature database [13]. It is exactly this syntactic characterization of the malicious code that makes standard misuse malware detectors so vulnerable to metamorphism. Thus, in order to handle metamorphism a malware detector should be able to recognize the metamorphic variants of a malware, namely the possible evolutions of the malicious code. The metamorphic engine is typically implemented as a set of code obfuscations that preserve program semantics to some extent. Thus, in order to handle metamorphism a malware detector should characterize infection in terms of semantic properties rather than syntactic properties (like signatures). For this reason researchers have started to investigate formal approaches to malware detection where infection is specified in terms of behavioral properties of programs (e.g., [1, 2, 6, 9]). As usual, the efficiency of these approaches is stated in terms of soundness (no false positives) and completeness (no false negatives) properties. In [6] the authors present a general framework based on program semantics and abstract interpretation for proving soundness and completeness of malware detectors in the presence of obfuscations. This semantic model for malware detection implicitly assumes that a malware appends its code and behavior to the one of the target program (the victim) without interacting with it. Hence, this formal model of malware infection is not appropriate for the description and identification of malware whose behaviors interferes with the one of the target program (either with spurious or real dependences added to obstruct program analysis).

In order to develop a more general theory that is able to describe the interactions between the malware and the target program we need a formal framework that is able to describe dependences between fragments of the same program. It is well known that non-interference [12] (NI) provides an ideal theory for reasoning on data dependencies in a program and that abstract non-interference consists in a generalization of the theory weakening the dependency analysis between data [7]. Our idea is to lift the ANI framework on programs and to define a sort of high-order ANI (HOANI) that characterizes dependences and relations among functions, and therefore programs, instead of data. The idea is that we detect infection when the semantics of a program matches the overall semantics of a target program corrupted by a malware. Indeed, if the malware detector could observe differences it would say that the specific malware has not infected the program, since we cannot recognize its semantics in the semantics of the program. This definition of malware detection allows to use HOANI for characterizing both soundness and completeness of the malware detectors, but allows even something more. We can inherits the attacker model and maximal information release characterizations of ANI, which lifted high order and instantiated to the malware detection field seem to provide a way to certify which classes of metamorphic engines do not deceive the detector, and to make a training of the detector depending on the metamorphic engine we aim to unveil. Finally, we prove that HOANI is a generalization of the semantic approach cited above [6] to malware detection since under specific conditions the two approaches collapse.

2 Background

Mathematical notation. If S and T are sets, then $\wp(S)$ denotes the powerset of S and $S \times T$ denotes the Cartesian product of S and T. If $f : S \longrightarrow T$, $Y \subseteq S$, and $X \subseteq T$ then $f(Y) \stackrel{\text{def}}{=} \{f(y) \mid y \in Y\}$ and $f^{-1}(X) \stackrel{\text{def}}{=} \left\{ x \,\middle|\, f(x) \in X \right\}$. We will often denote $f(\{x\})$ as $f(x)$ and use lambda notation for functions. $f \circ g \stackrel{\text{def}}{=} \lambda x.\ f(g(x))$. $\langle C, \leq \rangle$ denotes a poset C with ordering relation $\leq$, while $\langle C, \leq, \vee, \wedge, \top, \bot \rangle$ denotes a complete lattice C, with ordering $\leq$, *lub* $\vee$, *glb* $\wedge$, top and bottom element $\top$ and $\bot$ respectively. $\mathit{id} \stackrel{\text{def}}{=} \lambda x.\ x$ and $\mathbb{T} \stackrel{\text{def}}{=} \lambda x.\ \top$. The point-wise ordered set of monotone functions, denoted $C_1 \xrightarrow{\text{m}} C_2$, is a complete lattice $\langle C_1 \xrightarrow{\text{m}} C_2, \sqsubseteq, \sqcup, \sqcap, \mathbb{T}, \lambda x.\bot \rangle$. $f : C_1 \longrightarrow C_2$ is (completely) additive if f preserves *lub*'s of all subsets of C_1 (emptyset included). Continuity, denoted $\xrightarrow{\text{c}}$, holds when f preserved *lubs*'s of chains. Co-addittivity and co-continuity are dually defined.

Abstract interpretation. We use the framework of abstract interpretation [3, 4] for modeling both the observational capability of malware detector and the invariant properties of metamorphic engines. Abstract interpretation is used for reasoning on *properties* rather than on (concrete) data values. Abstract interpretation is a general theory for deriving sound approximations of the semantics of discrete dynamic systems, e.g., programming languages [3]. Approximation can be equivalently formulated either in terms of Galois connections or closure operators [4]. An *upper closure operator* (uco for short) $\rho : C \to C$ on a poset C, representing concrete objects, is monotone, idempotent, and extensive: $\forall x \in C.\ x \leq_C \rho(x)$. The upper closure operator is the function that maps the concrete values to their abstract properties, namely with the best possible approximation of the concrete value in the abstract domain. Formally, closure operators ρ are uniquely determined by the set of their fix-points $\rho(C)$, for instance $\mathit{Par} = \{\mathbb{Z}, \mathtt{ev}, \mathtt{od}, \varnothing\}$. For upper closures, $X \subseteq C$ is the set of fix-points of $\rho \in \mathit{uco}(C)$ iff X is a *Moore-family* of C, i.e., $X = \mathcal{M}(X) \overset{\text{def}}{=} \{\wedge S \mid S \subseteq X\}$ — where $\wedge\varnothing = \top \in \mathcal{M}(X)$. The set of all upper closure operators on C, denoted $\mathit{uco}(C)$, is isomorphic to the so called *lattice of abstract interpretations of* C [4]. If C is a complete lattice then $\mathit{uco}(C)$ ordered point-wise is also a complete lattice, $\langle \mathit{uco}(C), \sqsubseteq, \sqcup, \sqcap, \top, \mathit{id}\rangle$ where for every $\rho, \eta \in \mathit{uco}(C)$, $\{\rho_i\}_{i\in I} \subseteq \mathit{uco}(C)$ and $x \in C$: $\rho \sqsubseteq \eta$ iff $\forall y \in C.\ \rho(y) \leq \eta(y)$ iff $\eta(C) \subseteq \rho(C)$; $(\sqcap_{i\in I}\rho_i)(x) = \wedge_{i\in I}\rho_i(x)$; and $(\sqcup_{i\in I}\rho_i)(x) = x \Leftrightarrow \forall i \in I.\ \rho_i(x) = x$. Given an abstraction, we want also to understand whether the program computes *accurately* on the property. In general, the abstract interpretation framework guarantees that the abstract computation is *sound*, namely we can only lose information by computing on abstract properties. On the other hand, the accuracy of the computation is modeled in terms of *completeness*: an abstract domain is complete for a program if the computation of the program, on the abstract properties, corresponds precisely to the abstraction of the concrete computation. In other words, the abstract domain is as precise as possible with respect to the program to compute. The *best correct approximation* of f is $f^{bca} \overset{\text{def}}{=} \alpha \circ f \circ \gamma$ (or equivalently $\gamma \circ \alpha \circ f \circ \gamma \circ \alpha$). It is known that $f^\sharp$ is sound iff $f^{bca} \sqsubseteq f^\sharp$ and this implies that $\alpha(\mathit{lfp}(f)) \leq \mathit{lfp}(f^{bca}) \leq \mathit{lfp}(f^\sharp)$ [4]. In the following, if $[\![P]\!]$ is specified as fix-point of (a combination of) predicate-transformers $F_P : C \xrightarrow{c} C$, and $\rho \in \mathit{uco}(C)$, we denote by $[\![P]\!]^\rho$ the (fix-point) semantics associated with $F_P^{bca} = \rho \circ F_P \circ \rho$. $[\![P]\!]^\rho$ is the best correct abstract interpretation of P in ρ. In this case $\rho([\![P]\!]) \leq [\![P]\!]^\rho$.

Abstract non-interference. Abstract non-interference (ANI) [7] is a natural weakening of non-interference by abstract interpretation. Suppose the variables of program split in private ($\mathtt{H}$) and public ($\mathtt{L}$). Let $\eta, \rho \in \mathit{uco}(\mathbb{V}^{\mathtt{L}})$ and $\phi \in \mathit{uco}(\mathbb{V}^{\mathtt{H}})$, where $\mathbb{V}^{\mathtt{L}}$ and $\mathbb{V}^{\mathtt{H}}$ are the domains of $\mathtt{L}$ and $\mathtt{H}$ variables. η and ρ characterise the *attacker*. Let $\phi \in \mathit{uco}(\mathbb{V}^{\mathtt{H}})$, which states what, of the private data, can flow to the output observation, the so called *declassification* of ϕ [10]. A program P satisfies ANI, and we write $[\eta]P(\phi \Rightarrow \rho)$, if $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$:

$$\eta(l_1) = \eta(l_2) \ \wedge\ \phi(h_1) = \phi(h_2) \ \Rightarrow\ \rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}}). \quad (1)$$

This notion says that, whenever the attacker is able to observe the input property η and the output property ρ, then it can observe nothing more than the property ϕ of private input. It is possible to systematically characterize the most concrete output observation for a program, given the input one [7]. The idea is that of abstracting in the same object all the elements that, if distinguished, would generate a visible flow. On the other hand, we can characterize the maximal information that a program semantics allows to flow, namely which is the most abstract property that needs to be declassified in order to guarantee the non-interference of the program [7].

3 The Ingredients

Separability and Program Integration. Let us recall the notions of interleave and separability introduced in [11]. Consider two disjoint sets of variables $X = \{x_1 \dots x_n\}$ and $Y = \{y_1 \dots y_n\}$. We use notation $\bar{x}$ to refer to the tuple $\langle x_1 \dots x_n \rangle$, notation $\mathbf{x}_i$ to refer to the value stored in variable x_i, and notation $\bar{\mathbf{x}}$ to refer to the tuple of values $\langle \mathbf{x}_1 \dots \mathbf{x}_n \rangle$. We define the set of possible states over X and Y as follows:

$$\Sigma_{X:Y} = \left\{ \, \langle \mathbf{x}_1 \dots \mathbf{x}_n : \mathbf{y}_1 \dots \mathbf{y}_n \rangle \,\middle|\, X = \{x_1 \dots x_n\}, Y = \{y_1 \dots y_n\} \, \right\}$$

When $Y = \varnothing$ we refer to the set of states over X simply as Σ_X. Every trace $\sigma \in \Sigma^*_{X:Y}$ is of the form $\sigma = \langle \bar{\mathbf{x}}_1 : \bar{\mathbf{y}}_1 \rangle \langle \bar{\mathbf{x}}_2 : \bar{\mathbf{y}}_2 \rangle \dots$ with $\langle \bar{\mathbf{x}}_i, \bar{\mathbf{y}}_i \rangle \in \Sigma_{X:Y}$ for every i. Let ϵ denote the empty trace. We define the projection function $\pi_X : \Sigma_{X:Y} \rightarrow \Sigma_X$ as $\pi_X(\epsilon) = \epsilon$, $\pi_X(\langle \bar{\mathbf{x}}_1 : \bar{\mathbf{y}}_1 \rangle \sigma) = \bar{\mathbf{x}}_1 \pi_X(\sigma)$, and similarly the projection function $\pi_Y : \Sigma_{X:Y} \rightarrow \Sigma_Y$. According to [11] we define function $interleave : \Sigma^*_{X:Y} \times \Sigma^*_{X:Y} \rightarrow \Sigma^*_{X:Y}$ such that $interleave(\sigma_1, \sigma_2) = \sigma$ iff $\pi_X(\sigma) = \pi_X(\sigma_1)$ and $\pi_Y(\sigma) = \pi_Y(\sigma_2)$. A set of traces $\Gamma \in \Sigma_{X:Y}$ satisfies *separability* iff it is closed under interleave, namely if $\forall \sigma_1, \sigma_2 \in \Gamma$ then $interleave(\sigma_1, \sigma_2) \in \Gamma$.

We model program integration as a function $\mathfrak{I} : \mathbb{P} \times \mathbb{P} \longrightarrow \mathbb{P}$ that given two programs combines them into one. Let $\mathit{Var}(P)$ denote the variables of program P. We interpret the notions of interleaving and separability in the context of program integration.

Definition 1 *An integration function $\mathfrak{I} : \mathbb{P} \times \mathbb{P} \longrightarrow \mathbb{P}$ satisfies separability if for every pair of programs Q and T with disjoint variables, i.e., $\mathit{Var}(Q) \cap \mathit{Var}(T) = \varnothing$, the set of traces $[\![\mathfrak{I}(Q,T)]\!] \in \wp(\Sigma^*_{\mathit{Var}(Q):\mathit{Var}(T)})$ is closed for interleave.*

This means that, when the integration function satisfies separability, the behaviors of programs Q and T are kept separate and independent in the behavior of the integrated program $\mathfrak{I}(Q,T)$. In other words an integration functions satisfies separability when it does not add dependences between the programs it composes. Indeed, when we have separability we believe that it is reasonable to assume that the behavior of $\mathfrak{I}(Q,T)$ restricted to Q coincides exactly with the behaviour of Q, namely that if $\mathfrak{I}$ satisfies separability then $\forall Q, T \in \mathbb{P} : \; \pi_{\mathit{Var}(Q)}([\![\mathfrak{I}(Q,T)]\!]) = [\![Q]\!]$.

The Malware Detection Problem. A malware detector can be modeled as a function $D : \mathbb{P} \times \mathbb{P} \to \{\mathit{true}, \mathit{false}\}$ that decides whether a program is infected with a malware or a metamorphic variant of it. Given $M, P \in \mathbb{P}$ we denote with $M \hookrightarrow P$ the fact that program P is infected with malware M. An ideal metamorphic malware detector should be both *sound* (never erroneously claim that a program is infected) and *complete* (detect all metamorphic variants), namely it should satisfy the following:

$$D(M, P) = \mathit{true} \quad \Leftrightarrow \quad \exists M' \text{ metamorphic variant of } M : \; M' \hookrightarrow P$$

The weaker notions of relative soundness/completeness are used to certify soundness and completeness of a given malware detector wrt a class of obfuscations [6] .

Definition 2 *Let $\mathbb{O}$ be a set of obfuscations. A malware detector D is sound for $\mathbb{O}$ if $D(P, M) = \mathit{true} \;\Rightarrow\; \exists \mathcal{O} \in \mathbb{O} : \mathcal{O}(M) \hookrightarrow P$. A malware detector D is complete for $\mathbb{O}$ if $\forall\, \mathcal{O} \in \mathbb{O} : \; \mathcal{O}(M) \hookrightarrow P \;\Rightarrow\; D(P, M) = \mathit{true}$.*

In the following we formalize the notion of infection in terms of program integration: $M \hookrightarrow P$ iff $\exists\, T.\; [\![P]\!] = [\![\mathfrak{I}(M, T)]\!]$. Hence, the integration function $\mathfrak{I}$ models infection (we may have different infection functions). For instance, if the malware is a standard file infector appending its code to a target file, then the integration is simply the concatenation of the codes involved and it would be modeled by an integration function that satisfies separability.

Higher-order Abstract Noninterference. In order to model non-interference in *code transformations* such as code obfuscation and metamorphism, we consider an higher-order version of ANI, where the objects of observations are programs instead of values. Hence, we have a part of a program (semantics) that can change and that is not observable, and the environment which remains the same up to an observable property. The function analyzed by HOANI is a program integration function, which takes the two parts of the program and provides a program as result. The output observation is the best correct approximation of the resulting program. Consider programs $P \in \mathbb{P}$ and the corresponding semantics, i.e., $[\![P]\!]$. Hence, we define

$$\eta([\![P_1]\!]) = \eta([\![P_2]\!]) \wedge \; \phi([\![Q_1]\!]) = \phi([\![Q_2]\!]) \;\Rightarrow \rho([\![\mathfrak{I}(Q_1, P_1)]\!]) = \rho([\![\mathfrak{I}(Q_2, P_2)]\!]) \qquad (2)$$

Note that, the abstractions can be any abstract property on programs. In the following, we consider HOANI for a particular family of abstractions, and in particular for semantics' *bca*. In other words, if we have $\rho \in \mathit{uco}(\wp(\Sigma))$, then we consider $\rho^\rho \in \mathit{uco}(\wp(\Sigma) \xrightarrow{\text{m}} \wp(\Sigma))$ such that $\rho^\rho \overset{\text{def}}{=} \lambda f.\; \rho f \rho$ [5]. By noting that, $\rho^\rho([\![P]\!]) = [\![P]\!]^\rho$ (defined in Sect. 2), we can rewrite Eq. 2 in the following HOANI notion:

$$[\![P_1]\!]^\eta = [\![P_2]\!]^\eta \wedge \; [\![Q_1]\!]^\phi = [\![Q_2]\!]^\phi \implies [\![\mathfrak{I}(Q_1, P_1)]\!]^\rho = [\![\mathfrak{I}(Q_2, P_2)]\!]^\rho \qquad (3)$$

Example 1. Consider the following program fragments, where $10! = 3628800$:

$$Q_1 : \left[\begin{array}{l} prod = 1; x = 1; \\ \textbf{while } x < 11 \, \{ \\ \qquad prod = prod \cdot x; \\ \qquad x = x + 1\}; \end{array}\right. \qquad Q_2 : \left[\begin{array}{l} prod = 10!; x = 11; \\ \textbf{while } x > 1 \, \{ \\ \qquad x = x - 1; \\ \qquad prod = prod / x\}; \end{array}\right.$$

$$P_1 : \left[\begin{array}{l} sum = 0; x = 1; \\ \textbf{while } x < 11 \, \{ \\ \qquad sum = sum + x; \\ \qquad x = x + 1\}; \end{array}\right. \qquad P_2 : \left[\begin{array}{l} sum = 55; x = 11; \\ \textbf{while } x > 1 \, \{ \\ \qquad x = x - 1; \\ \qquad sum = sum - x\}; \end{array}\right.$$

Consider, as $\mathfrak{I}$ ($\mathcal{T} = [\![\mathfrak{I}]\!]$) the integrating algorithm proposed by [8], providing the following resulting programs:

$$\mathfrak{I}(Q_1, P_1) : \left[\begin{array}{l} prod = 1; sum = 0; \\ x = 1; \\ \textbf{while } x < 11 \, \{ \\ \qquad prod = prod \cdot x; \\ \qquad sum = sum + x; \\ \qquad x = x + 1\}; \end{array}\right. \qquad \mathfrak{I}(Q_2, P_2) : \left[\begin{array}{l} prod = 10!; sum = 55; \\ x = 11; \\ \textbf{while } x > 1 \, \{ \\ \qquad x = x - 1; \\ \qquad prod = prod / x; \\ \qquad sum = sum - x\}; \end{array}\right.$$

Consider the abstract domain $\iota \in uco(\wp([-\mathfrak{m}, \mathfrak{m}]))$ of limited intervals, where $\mathfrak{m} \in \mathbb{Z}$ is the maximal integer. In this case $\iota(x) = [\min(x), \max(x)]$. Interval analysis is defined in [3], with standard *bca* abstract interpretations for arithmetic operations on intervals: $\odot$, $\oplus$, $\ominus$. Then we have that

$$[\![Q_1]\!]^\iota = [\![Q_2]\!]^\iota \;\wedge\; [\![P_1]\!]^\iota = [\![P_2]\!]^\iota \implies [\![\mathfrak{I}(Q_1, P_1)]\!]^\iota = [\![\mathfrak{I}(Q_2, P_2)]\!]^\iota$$

This HOANI property of the considered integration algorithm tells us that we can vary the involved programs leaving unchanged the variables' intervals without inducing any variation in the interval analysis of the resulting program.

4 Malware detection by unveiling program dependencies

4.1 Abstract noninterference-based malware detector

In this section, we define a notion of malware detector inspired by higher order abstract noninterference, let us call it ANIMD. The idea is that a program P is infected with a possibly metamorphic variant of malware M if it is (semantically) equivalent, at least up to an observation (program analysis), to the integration of a code segment T with the code of the malware M. Formally, given $\rho \in uco(\wp(\Sigma^*_{\mathsf{Var}(P)}))$:

$$\text{ANIMD}_\rho(M, P) = true \;\;\Leftrightarrow\;\; \exists T \in \mathbb{P} : [\![\mathfrak{I}(M, T)]\!]^\rho = [\![P]\!]^\rho$$

Namely a program P is infected with a malware M if it behaves wrt ρ like a target program T infected with malware M.

Given a metamorphic engine ME we assume that it is possible to identify a semantic property ϕ that is preserved by any code transformation generated by ME, while each transformation changing ϕ cannot be generated by ME. This means that ME can be modeled as a semantic property $\phi \in uco(\wp(\Sigma^*_{\mathsf{Var}(M)}))$ and that the set of all the obfuscating transformations generated by ME can be formalized as follows:

$$\mathbb{O}_\phi = \left\{ \mathcal{O} \,\middle|\, \forall P, Q \in \mathbb{P}.\, [\![P]\!]^\phi = [\![Q]\!]^\phi \;\Leftrightarrow\; P = \mathcal{O}(Q) \right\}.$$

This are exactly all and only the transformations used by the malware equipped with ME as stealthing technique. We can either assume to know this property, or given a set of metamorphic malware variants we can derive it and then use it to model the metamorphic engine (possibly catching also unseen variants). First of all, let us rewrite HOANI in the context of malware detector: by changing the version of the malware, up to an observable property ϕ, the malware detector analysing ρ is not deceived by the differences between the versions and recognize the same infection in both the analyzed programs. Hence, we define HOANI^ϕ_ρ:

$$[\![M]\!]^\phi = [\![M']\!]^\phi \implies [\![\mathfrak{I}(M,P)]\!]^\rho = [\![\mathfrak{I}(M',P)]\!]^\rho \tag{4}$$

At this point we study the precision of the malware detectors based on HOANI in terms of soundness and completeness.

Theorem 2 (Soundness). *Let* $\mathbb{O}_\phi = \left\{ \mathcal{O} \,\middle|\, \forall P, Q \in \mathbb{P}.\, [\![P]\!]^\phi = [\![Q]\!]^\phi \;\Leftrightarrow\; P = \mathcal{O}(Q) \right\}$, *then* ANIMD_ρ *is sound for* $\mathbb{O}_\phi$ *whenever:*

$$[\![M]\!]^\phi = [\![M']\!]^\phi \impliedby [\![\mathfrak{I}(M,P)]\!]^\rho = [\![\mathfrak{I}(M',P)]\!]^\rho \tag{5}$$

Theorem 3 (Completeness). *If* HOANI^ϕ_ρ *holds, then* ANIMD_ρ *is complete for* $\mathbb{O}_\phi$.

4.2 Certifying and Training Malware Detectors.

In this section we discuss how we can exploit the HOANI framework in order to understand how we can *certify* the "power" of a malware detector in terms of the classes of metamorphic engines unable to deceive it, and how we can do a *training* of malware detectors starting from a class of obfuscation techniques characterizing a metamorphic engine that we aim to defeat. In this way we could formally understand the relation between the metamorphic invariant property and the analysis performed by the detector.
The ANI framework allows to describe two transformations, one characterizing the most concrete output observation unable to observe interferences, and the other characterizing the maximal property that do not cause interference [7]. We believe that these two transformations, once lifted high order, would provide exactly a way for certifying and training malware detectors. The main difference between ANI and HOANI is that while abstracting data means to consider properties of data, i.e., sets of values; abstracting programs means to consider the best correct approximation of their semantics,

i.e., the abstraction of a function is a more abstract function instead of a set of functions. This difference makes not so immediate to extend ANI from data to functions and requires a deeper analysis of a higher-order notion of abstract non-interference. Note that, because the domain transformers introduced for ANI [7] extended to the definition above of HOANI would generate sets of programs and therefore of semantics (i.e., functions), which in general represent program/semantics properties, we can build a correspondence between semantic properties, i.e., sets of semantic functions, and best correct approximations. In other words, we can always associate a best correct approximation with any set of functions, while we can construct a set of functions corresponding to any given best correct approximation of a given function.
Certification: Given ρ in HOANI we can characterize the maximal amount of information released ϕ. This property ϕ is non-redundant, i.e., any change of ϕ do cause interference, and it is such that when it is invariant then there is no interference in the observation ρ. Hence, if we start from a malware detector ANIMD_ρ, we can characterize the most concrete property ϕ such that ANIMD_ρ is sound and complete for $\mathbb{O}_\phi$. This means that ANIMD_ρ is sound and complete for any metamorphic engine whose code transformations preserves at least ϕ
Training: Given a property ϕ the HOANI framework allows to characterize the most concrete observation unable to observe interferences when the property ϕ is unchanged. In other words, if we start from a set of obfuscations $\mathbb{O}$, whose semantic invariant is the property ϕ then we can characterize the most concrete ρ such that the corresponding ANIMD_ρ is complete for $\mathbb{O}$. Namely, we can modify the observation capability of the malware detector depending in the class of obfuscation we aim to defeat.

4.3 What's new in ANIMD?

In this section we compare the prosed ANIMD with the closest framework of semantic-based malware detectors based on abstract interpretation [6]. The two approaches are clearly related since both model the malware detector as parametric on the program analysis that it is able to perform. Moreover, in both the approaches the malware code has in some way to be separated by the original program and both the approaches characterize classes of obfuscation techniques, those used by a metamorphic engine, in terms of the invariant property left unchanged by the transformations. This means that we can quite easily compare these two approaches. In particular, we show that ANIMD generalize all these aspects by considering the best correct approximation of the program semantics instead of the output abstraction, and by considering a generic integration function instead of the trivial composition of programs. Hence, let us first recall the basic definitions of the first semantic malware detector [6].

Semantic Malware Detector. The idea of [6] is to classify a program P as infected by a possibly metamorphic variant of malware M if there exists a portion of P whose abstract behavior corresponds to the abstract behavior of M. This implicitly assumes that infection does not add dependences between the malware and the target program, namely that the integration function that models infection satisfies separability. Given $\rho \in uco(\wp(\Sigma_{\mathsf{Var}(M)}))$, we can rewrite the semantic malware detector of [6] as:

$$\mathrm{SMD}_\rho(M, P) = \mathit{true} \quad \Leftrightarrow \; \exists Q, T \in \mathbb{P} : [\![P]\!] = [\![\Im(Q, T)]\!] \wedge \rho([\![M]\!]) = \rho([\![Q]\!])$$

SMD vs ANIMD. Observe that SMD_ρ decides infection by comparing the abstraction of the concrete semantics of programs, i.e., $\rho([\![M]\!]) = \rho([\![Q]\!])$, while ANIMD_ρ decides infection by comparing the abstract semantics of programs, i.e., $[\![\Im(M, T)]\!]^\rho = [\![P]\!]^\rho$. The abstraction of the concrete semantics and the abstract computation of the semantics collapse when the abstract domain ρ is complete for the computation of program semantics as shown by the following result.

Lemma 4. *If f is complete for ρ, i.e., $\rho \circ f = \rho \circ f \circ \rho$ then we can apply the fix point Kleene transfer, namely $\rho\, \mathit{lfp} f = \mathit{lfp}\; \rho \circ f \circ \rho$.*

Thus, in order to compare SMD_ρ and ANIMD_ρ we have to assume the completeness of the domain ρ for the semantic computation, i.e., $\forall P \in \mathbb{P} : \rho([\![P]\!]) = \rho(\mathit{lfp} F_P) = \mathit{lfp} \rho \circ F_P \circ \rho = [\![P]\!]^\rho$.

Another difference between SMD and ANIMD is given by the computational domain that they consider: SMD observes properties of the behaviour of the malware, while ANIMD properties of the behaviour of the whole infected program. Thus, in order to understand their relation we define the following domain extension: Given $\rho \in uco(\wp(\Sigma^*_{\mathsf{Var}(M)}))$ we denote $\widetilde{\rho} \in uco(\wp(\Sigma^*_{\mathsf{Var}(M)})) \times uco(\wp(\Sigma^*_{\mathsf{Var}(T)}))$ any abstraction that is ρ on $\mathsf{Var}(M)$, i.e., $\widetilde{\rho} = \rho \times \eta$ where $\eta \in uco(\wp(\Sigma^*_{\mathsf{Var}(T)}))$.

Theorem 5. *Consider an integration function $\Im$ that satisfies separability, two abstract domains ρ and $\widetilde{\rho}$ that are complete for the computation of program semantics and assume that Equation* 4 *and Equation* 5 *hold, then* $\mathrm{SMD}_\rho(M, P) \Leftrightarrow \mathrm{ANIMD}_{\widetilde{\rho}}(M, P)$.

5 Conclusions and future works

In this work we have begun to investigate the possibility of exploiting the ANI theory for detecting malware infection. To this end we have started to reason on an high order version of the standard ANI framework that allows to reason on dependences and interferences among programs (instead of data). We have formalized the malware detection problem in terms of HOANI and we have proved that the malware detector ANIMD based on HOANI generalizes the semantic malware detector SMD proposed in [6]. An interesting feature of ANIMD is that it is parametric on the infection strategy used by the malware and that it can model possible interactions between the malware and the target program. Another reason that makes our approach promising is the possibility to develop systematic techniques for certifying and training malware detectors. This can be done by lifting high order the ANI transformations that characterize respectively the most concrete output observation unable to detect interferences, and the maximal property that do not cause interference. Indeed, the ability of certifying the precision of a given malware detector, and the possibility of deriving the best malware detector wrt a metamorphic engine are two important challenges in the battle against metamorphic malware. To this end we have to deeply understand and develop the HOANI theory beyond ANIMD.

Based on these results, our goal is to develop a systematic strategy for the design of the best malware detector for a given class of metamorphic code variants. To this end we first need to develop a technique for learning the metamorphic engine ME that has generated the considered malware variants. Next we have to characterize the invariant property ϕ of all the generated variants in order to derive the observation property ρ that characterizes detection for ANIMD_ρ. We believe that this theoretical identification of the program property ρ that the malware detector should consider in order to handle metamorphism for ME can given useful insight in the design and implementation of a malware detector tool able to defeat ME.

References

1. P. Beaucamps, I. Gnaedig, and J. Y. Marion. Behavior abstraction in malware analysis. In *Runtime Verification (RV'10)*, volume 6418 of *LNCS*, pages 168–182, 2010.
2. M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-aware malware detection. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'05)*, pages 32–46, Oakland, CA, USA, 2005.
3. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of Conf. Record of the 4th ACM Symp. on Principles of Programming Languages* (*POPL '77*), pages 238–252, New York, 1977. ACM Press.
4. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of Conf. Record of the 6th ACM Symp. on Principles of Programming Languages* (*POPL '79*), pages 269–282, New York, 1979. ACM Press.
5. P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages) (Invited Paper). In *Proc. of the 1994 IEEE Internat. Conf. on Computer Languages* (*ICCL '94*), pages 95–112, Los Alamitos, Calif., 1994. IEEE Comp. Soc. Press.
6. M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. A semantics-based approach to malware detection. *ACM Trans. Program. Lang. Syst.*, 30(5):1–54, 2008.
7. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL '04)*, pages 186–197, New York, 2004. ACM-Press.
8. S. Horwitz, J. Prins, and T. Reps. Integrating noninterfering versions of programs. *ACM Trans. Program. Lang. Syst.*, 11(3):345 – 387, 1989.

9. J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith. Detecting malicious code by model checking. In *Proceedings of the 2nd International Conference on Intrusion and Malware Detection and Vulnerability Assessment (DIMVA'05)*, volume 3548 of *LNCS*, pages 174–187, 2005.
10. I. Mastroeni. On the rôle of abstract non-interference in language-based security. In K. Yi, editor, *Third Asian Symp. on Programming Languages and Systems (APLAS '05)*, volume 3780 of *Lecture Notes in Computer Science*, pages 418–433, Berlin, 2005. Springer-Verlag.
11. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium in Security and Privacy*, pages 79–93, Los Alamitos, Calif., 1994. IEEE Comp. Soc. Press.
12. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE J. on selected ares in communications*, 21(1):5–19, 2003.
13. P. Ször. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, Boston, MA, USA, 2005.