Hunting distributed malware with the κ-calculus

Mila Dalla Preda[1] and Cinzia Di Giusto[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
[2] INRIA Rhône Alpes, Grenoble, France

Abstract. The defense of computer systems from malicious software attacks, such as viruses and worms, is a key aspect of computer security. The analogy between malicious software and biological infections suggested us to use the κ-calculus, a formalism originally developed for the analysis of biological systems, for the formalization and analysis of malicious software. By modeling the different actors involved in a malicious code attack in the κ-calculus and by simulating their behavior, it is possible to extract important information that can drive in the choice of the defense technique to apply.

1 Introduction

The challenge. According to Grimes, "Malicious code is any software program designed to move from computer to computer and network to network in order to intentionally modify computer systems without the consent of the owner" [16]. Malicious codes are classified according to their key features: the ability to propagate and the potential to perform a damage (or *payload*) [21]. The most popular classes of malicious code include viruses, worms, Trojan horses, spyware, trap doors and logic bombs. In general, the term *malware* refers to a malicious code regardless of its classification.

The enlarging size and complexity of modern information systems, together with the growing connectivity of computers through the Internet, have facilitated the spread of malware [21]. The past thirty years have seen a continuous growth of the threats of malware both in the danger and in the complexity of the malicious codes. Nowadays, one of the most sophisticated techniques used by hackers is to exploit the resources of some victim machines in order to create and control powerful networks of compromised nodes called *botnets*. A botnet consists of a collection of victim machines running specific programs, called bots, that are remotely controlled by the attacker, called botmaster [20]. Typical applications of botnets are the implementation of distributed denial of service attacks and of e-mail spamming. Botnets represent a dangerous, potent and quick evolving threat that is yet to be fully understood. Their potential resides in distributing the maliciousness over the network. This makes detection much harder and poses a global threat that can count on the cooperation of many machines.

Since the threat of malware attacks is an unavoidable problem in computer security, it is crucial to develop both sophisticated models for expressing and understanding distributed malware, i.e. botnets, and efficient techniques of defense from them. A standard defense technique is misuse malware detection which consists in detecting malicious code by scanning for predefined sequences of bytes that act like fingerprints. A novel promising approach regards the monitoring of malware propagation in a network both for studying properties of the considered malware and/or for finding ways to immunize the network. While misuse malware detection is based on an abstract representation of the inner working of a malicious code viewed as a single entity, the propagation monitoring approach is based on the characterization of the interactions of a malicious code with the environment. Indeed, the monitoring of malware propagation turns out to be particularly interesting for analyzing the topology of botnets and for understanding, detecting and stopping bot propagation [9].

The challenging behavior of malware suggests the development of a formal framework for the specification and analysis of malicious software. The standard formal definitions of viruses by Cohen [7] and Adleman [1] are not appropriate for modeling and analyzing the behavior and the propagation of modern distributed malware. Cohen characterizes viruses in terms of their ability to replicate with respect to a given Turing machine, while Adleman provides a more abstract definition of computer viruses based on recursive functions where an infected program either *imitates* the behavior of the original program, *injures* the machine or *propagates* infection. Interesting generalizations of the Adleman's definition are the ones given by Zou and Zhou [26] and Bonfante et al. [4]. All these abstract theoretical models based on Turing machines and recursive functions are used to prove that the malware detection problem is undecidable.

Observe that all these models, based on Turing machines or recursive functions, are not the most appropriate ones for describing the interaction and the cooperation between different systems. In fact, modern malware as botnets cannot be described and specified in a convenient and detailed way with the standard theoretical models. The advent of these interaction-based malware requires the development of new formal models for malware analysis. We mention here the theoretical model known as k-ary malware that distributes the malicious behavior into many concurrent codes [14]. At the moment there exist only trivial implementations of k-ary malware, but in principle, they could avert standard malware detection strategies. Another related work is the recent one of Jacob et al. [17], where the authors propose a model of malware based on the Join calculus. They found their definition of malware on the notion of virus given by Cohen [7] and extend it to support concurrent interactions. They focus on the malware detection problem and prove that in general it is undecidable in the Join calculus. Furthermore, they identify a fragment of the Join calculus where this problem becomes decidable even if it is not clear whether real malware can be modeled in the identified fragment.

The idea. The main objective of this work is to design a model for the specification and analysis of standard and distributed malware together with their propagation mechanisms. This accounts to consider formalisms able to express the concurrent interaction between the multiple actors involved in a malicious attack. When dealing with concurrency, languages such as CCS [22], ACP [3] and the π-calculus [24], are among the most natural choices. This is also the reason why those languages have largely inspired the design of calculi for the biological world. In this setting, where not only concurrent interactions but also spatial aspects such as containment or relative position (i.e. topology) are essential, new formal frameworks like brane calculus [5], Ambients [6] and κ-calculus [12] have been proposed. As biological virus are generally associated to malware, we find it natural to look for a suitable calculus among the ones designed for computational biology.

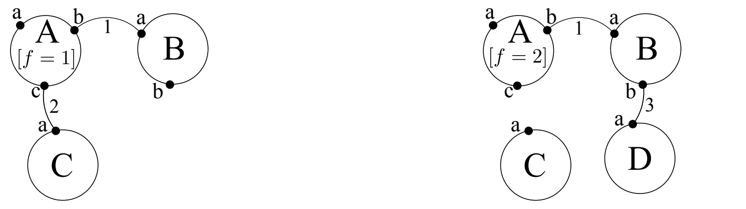
We propose to use the κ-calculus to model distributed malicious attacks. The κ-calculus is a language for the specification of systems that can be represented as graphs where nodes are equipped with an internal state and labeled ports, and edges specify connections between labeled ports of nodes. Rewriting rules associated to each system model the graph evolution. We believe that the κ-calculus is suitable for assessing the malware threat, also in the case of distributed malware, for the following reasons:

1. κ allows a detailed specification of the mechanisms of interactions between concurrent systems. One of the main advantages of using κ with respect to other process calculi as the π-calculus or the Join calculus, resides in its graphical nature that allows the description of spatial properties. Moreover, the set of rules defining the behavior of malware can be isolated and distinguished from the rules describing the normal evolution of a system.
2. κ offers a framework where the granularity of the model can be chosen depending on the specific problem to represent, i.e., one can model programs inside a single machine, or machines abstracting from the programs they are running.
3. Recent works have shown how topology plays a fundamental role in the analysis of malware propagation [15]. κ-systems are essentially graphs and this allows a straightforward definition of different network topologies. Moreover, since κ is a rewriting model, it is suitable for the specification of networks whose topology evolves over time as in [23].
4. The analysis of malware formalized in κ can exploit the tools for the simulation of κ-systems [8]. Moreover, these tools – developed in the biological setting – allow the specification of stochastic measures that can model the probability of a certain node to be infected, the rate of propagation of a given infection, etc.

In Section 2 we briefly introduce the κ-calculus. Section 3 formalizes the notion of malware with respect to an environment and shows how the proposed definition can be instantiated to the different kinds of standard and distributed malware. In Section 4 we present and discuss how the proposed model can be employed to study the propagation strategies of specific malware. Section 5 concludes by discussing future developments.

2 The κ-calculus

In this section we recall the basic definitions of the κ-calculus [12] independently from its biological domain of application. Some terminology has been changed and some constraints (like the separation between destruction and construction rules, typical of biological settings) have been relaxed. The κ-calculus is a graph rewriting model where objects are represented by nodes and their relationships by edges. More formally, let us consider the following countable sets: `Nodes` with elements $A, B, C, \ldots$; `Gates` with elements $a, b, c \ldots$; `Arms` with elements $1, 2, 3, \ldots$; `Fields` with elements $f_1, f_2, \ldots$. Let $Arms(A) \subseteq$ `Arms`, $Gates(A) \subseteq$ `Gates` and $Fields(A) \subseteq$ `Fields` denote, respectively, the set of arms, gates and fields of a node A. A *state* α is a map



(a) The start up condition of the system (b) The system after the evolution

Fig. 1. Graphical representation of the system $A[f = 1](a = \varepsilon, b = 1, c = 2), B(a = 1, b = \varepsilon), C(a = 2)$ and the application of rule $A[f = 1](c = 2), B(b = \varepsilon), C(a = 2) \to A[f = 2](c = \varepsilon), B(b = 3), C(a = \varepsilon), D(a = 3)$.

from `Fields` to a finite set of possible values. An *interface* $\mathcal{S}$ is a map `Gates` $\mapsto$ `Arms` $\cup \{\varepsilon\}$ where $\mathcal{S}(a) = \varepsilon$ means that no arm is associated to gate a.

The domain of a partial function $f : D \mapsto C$ is $dom(f) = \{d \in D \mid f(d) \neq \bot\}$. The *union* of two partial functions $f, g : D \mapsto C$ such that $dom(f) \cap dom(g) = \emptyset$, is a partial function $f + g : D \mapsto C$ with domain $dom(f + g) = dom(f) \cup dom(g)$ such that $(f + g)(d) = f(d)$ if $d \in dom(f)$, $(f + g)(d) = g(d)$ if $d \in dom(g)$, and $f + g(d) = \bot$ otherwise. $f \leq g$ if $dom(f) \subseteq dom(g)$ and $\forall x \in dom(g) : f(x) = g(x)$ or $f(x)$ undefined.

A component is a node A with a state that specifies all the values of the fields of A, and an interface that specifies which gates of A are connected to some arms (edges).

Definition 1. *A* component $A[\alpha](\mathcal{S})$ *is a node A equipped with a state map α and an interface map $\mathcal{S}$ that are total w.r.t. A: $dom(\alpha) = Fields(A)$ and $dom(\mathcal{S}) = Gates(A)$.*

If a component has no fields we omit the part [ ]. A system is a multiset of components and it can be graphically represented as a non-oriented graph with nodes denoting components and edges modeling arms (see Fig. 1(a) for an example).

Definition 2. *A* system Σ *is defined as $\Sigma ::= A[\alpha](\mathcal{S}) \mid \Sigma, \Sigma$, where "," denotes the associative operator for composition and each arm occurs exactly twice.*

We speak about component-projection when we have a node with a partial specification of its state and interface.This notion can be naturally extended to systems by saying that a system-projection is a collection of component-projections. If we consider the system in Fig. 1(a), $B(a^1)$ is a component-projection of $B(a^1 + b^\varepsilon)$.

Definition 3. $A[\alpha](\mathcal{S})$ *is a* component-projection *(of A) if α and $\mathcal{S}$ are not total with respect to A i.e., $dom(\alpha) \subseteq Fields(A)$ and $dom(\mathcal{S}) \subseteq Gates(A)$.*

$A_1[\alpha_1](\mathcal{S}_1) \ldots A_n[\alpha_n](\mathcal{S}_n)$ *is a* system-projection *(of $A_1 \ldots A_n$) if for every $i \in [1..n]$ we have that $A_i[\alpha_i](\mathcal{S}_i)$ is a component projection (of A_i) and every arm occurs exactly twice.*

Definition 4. *A* rule *has the form $L \to R$ where the left hand side L and the right hand side R are system-projections:*

$$\begin{aligned} A_1[\alpha_1](\mathcal{S}_1), \ldots, A_n[\alpha_n](\mathcal{S}_n) \to\ & A_{i_1}[\alpha'_{i_1}](\mathcal{S}'_{i_1}), \ldots, A_{i_m}[\alpha'_{i_m}](\mathcal{S}'_{i_m}), \\ & B_1[\beta_1](\mathcal{F}_1), \ldots, B_k[\beta_k](\mathcal{F}_k) \end{aligned}$$

where for all $j \in [1..m]$: $i_j \in J \subseteq \{1 \ldots n\}$, $dom(\alpha_{i_j}) = dom(\alpha'_{i_j})$ and $\mathcal{S}'_{i_j} \leq \mathcal{S}_{i_j}$, for every $i \in \{1 \ldots n\} \setminus J$ we have that α_i and $\mathcal{S}_i$ are total with respect to A_i and for every $l \in [1..k]$ we have that β_l and $\mathcal{F}_l$ are total with respect to B_l.

The intuition is that a rule can have the following effects: (i) modify the state of the given components by changing the values of the already specified fields; (ii) add new arms to the gates that the given components map to ε; (iii) add new components (i.e. fully specified terms), (iv) delete arms of given components by modifying the interface in such a way that the corresponding gate maps to ε; (v) remove the elements of the system-projection only if they are components (i.e., they are fully specified). As an example see Fig. 1(b).

Definition 5. *The* structural equivalence *relation between systems, denoted $\equiv$, is the least equivalence satisfying:*

- *$\Sigma, \Pi \equiv \Pi, \Sigma$;*
- *$\Sigma \equiv \Pi$ if there exists an injective renaming ι on arms such that Σ is obtained from Π by renaming its arms according to ι, denoted $\Sigma = \iota(\Pi)$.*

Two component-projections $A[\alpha_1](\mathcal{S}_1)$ and $A[\alpha_2](\mathcal{S}_2)$ on the same node A are *disjoint* when $dom(\alpha_1) \cap dom(\alpha_2) = \emptyset$ and $dom(\mathcal{S}_1) \cap dom(\mathcal{S}_2) = \emptyset$. We define the *sum* of disjoint component-projections as: $A[\alpha_1](\mathcal{S}_1) \oplus A[\alpha_2](\mathcal{S}_2) = A[\alpha_1 + \alpha_2](\mathcal{S}_1 + \mathcal{S}_2)$. This can be extended to systems in the expected way. A component-projection, as well as a system-projection, can be instantiated to a set of possible components, or systems, by specifying the values of the undefined fields and the missing gate-arm relations. The notion of projection allows us to define the reduction relation for the κ-calculus.

Definition 6. *Let us consider a set $\mathbb{R}$ of rules and the systems $\Sigma, \Sigma', \Pi, \Pi', \Gamma$. The* reduction relation *associated to $\mathbb{R}$, denoted $\rightarrow_{\mathbb{R}}$, is the least relation that satisfies the followings:*

- *if $L \rightarrow R \in \mathbb{R}$, $\Sigma = L \oplus \Delta$, and $\Pi = R \oplus \Delta$ then: $\Sigma \rightarrow_{\mathbb{R}} \Pi$;*
- *if $\Sigma \rightarrow_{\mathbb{R}} \Pi$ and $(Arms(\Pi) \setminus Arms(\Sigma)) \cap Arms(\Gamma) = \emptyset$ then $\Sigma, \Gamma \rightarrow_{\mathbb{R}} \Pi, \Gamma$;*
- *if $\Sigma \equiv \Sigma'$, $\Sigma' \rightarrow_{\mathbb{R}} \Pi'$ and $\Pi' \equiv \Pi$, then $\Sigma \rightarrow_{\mathbb{R}} \Pi$.*

The first condition specifies how to apply rule $L \rightarrow R$ to a system Σ with a system-projection that matches L. The second condition states that rules can be applied to portions of the system only if the arms created by the rule are not present in the rest of the graph. The third condition says that the reduction relation between systems can be extended to structurally equivalent systems. We denote with $\Rightarrow$ the reflexive and transitive closure of $\rightarrow$. Given two systems Σ, Π with the respective sets of rules $\mathbb{R}_\Sigma, \mathbb{R}_\Pi$, we use notation $(\Sigma, \mathbb{R}_\Sigma), (\Pi, \mathbb{R}_\Pi) \rightarrow (\Sigma', \mathbb{R}_\Sigma), (\Pi', \mathbb{R}_\Pi)$ to refer to $\Sigma, \Pi \rightarrow_{\mathbb{R}_\Sigma \cup \mathbb{R}_\Pi} \Sigma', \Pi'$. Moreover, we denote with $\Sigma \overset{\mathbb{R}}{\Longrightarrow} \Sigma'$ the fact that during the evolution we have used only the rules in the set $\mathbb{R}$.

3 Modeling Malware in κ

Since this is the first time that the κ-calculus is used for modeling programs, we need to formally specify what is a program and what is an execution environment in the κ-framework.

Definition 7. *A* program $\mathtt{P}$ *in* κ *is a pair* $(\Sigma_{\mathtt{P}}, \mathbb{R}_{\mathtt{P}})$*, where* $\Sigma_{\mathtt{P}}$ *is a system and* $\mathbb{R}_{\mathtt{P}}$ *is the set of rules that describe the behaviour of* $\mathtt{P}$ *in the environment. Let Progr denote the set of programs in* κ*. An* environment $\mathcal{E}$ *is a collection of programs* $\mathtt{P}_1, \ldots, \mathtt{P}_n$ *with* $\mathtt{P}_i \in$ *Progr for all* $i \in [1, n]$.

We denote with $\mathcal{E}[\mathtt{P} \leftarrow \mathtt{P}']$, the environment $\mathcal{E}$ where program $\mathtt{P}$ has been replaced with program $\mathtt{P}'$. We extend the notion of structural equivalence to programs by saying that two programs $\mathtt{P} = (\Sigma_{\mathtt{P}}, \mathbb{R}_{\mathtt{P}})$ and $\mathtt{Q} = (\Sigma_{\mathtt{Q}}, \mathbb{R}_{\mathtt{Q}})$ are *structurally equivalent*, denoted $\mathtt{P} \doteqdot \mathtt{Q}$, if they have systems that are structurally equivalent and the same set of rules, i.e., $\mathtt{P} \doteqdot \mathtt{Q}$ if $\Sigma_{\mathtt{P}} \equiv \Sigma_{\mathtt{Q}}$ and $\mathbb{R}_{\mathtt{P}} = \mathbb{R}_{\mathtt{Q}}$ up to renaming. Thus, programs that are structurally equivalent cause structurally equivalent evolutions on structurally equivalent systems. We can now specify a formal definition of malware in the κ-calculus, where a malware is a program that behaves in a specific way in an environment. The proposed definition is inspired by the ones of Cohen [7] and Adleman [1]. In fact, according to Cohen, we provide a notion of malware with respect to a given environment and, according to Adleman, our notion of malware specifies the possible behaviors of the malicious code either as imitate, injure and infect. Moreover, by modeling the malicious behaviors in the κ-framework we can specify the interactions of the malware with the environment and we can describe the behavior of distributed malware.

Definition 8. *A program* $\mathtt{M} = (\Sigma_{\mathtt{M}}, \mathbb{R}_{\mathtt{M}})$ *is a* malware w.r.t. environment $\mathcal{E}$ *if it behaves in one of the following ways in environment* $\mathcal{E}$*:*

- Imitate*: If* $\mathtt{M}$ *assumes the presence of a host program then it has the following structure* $\mathtt{M} = h(\mathtt{MC}, \mathtt{Q})$ *where* $\mathtt{MC}$ *is the malicious code,* $\mathtt{Q} = (\Sigma_{\mathtt{Q}}, \mathbb{R}_{\mathtt{Q}})$ *is the program hosting it, and* h : *Progr* $\times$ *Progr* $\rightarrow$ *Progr is the infection function that given* $\mathtt{MC}$ *and* $\mathtt{Q}$ *returns the program* $h(\mathtt{MC}, \mathtt{Q})$ *obtained by infecting* $\mathtt{Q}$ *with* $\mathtt{MC}$*. In this case we say that program* $h(\mathtt{MC}, \mathtt{Q})$ *is able to imitate the behavior of the host program* $\mathtt{Q}$ *if the following holds: if* $\mathtt{Q}, \mathcal{E} \overset{\mathbb{R}_{\mathtt{Q}}}{\Longrightarrow} \mathtt{Q}', \mathcal{E}'$ *then* $h(\mathtt{MC}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_{\mathtt{Q}}}{\Longrightarrow} \mathtt{Im}(\mathcal{E}; \mathtt{Q})$ *and* $\mathtt{Im}(\mathcal{E}, \mathtt{Q}) = h(\mathtt{MC}, \mathtt{Q}'), \mathcal{E}''$ *such that* $\mathcal{E}'' \doteqdot \mathcal{E}'$*.*
- Injure: $\mathtt{M}$ *performs the intended payload on* $\mathcal{E}$*:* $\mathtt{M}, \mathcal{E} \overset{\mathbb{R}_{\mathtt{M}}}{\Longrightarrow} \mathtt{M}', \mathtt{Pl}(\mathcal{E}; \mathtt{M})$*, where* $\mathtt{Pl} : \wp(\mathit{Progr}) \times \mathit{Progr} \rightarrow \wp(\mathit{Progr})$ *is the payload function that, given an environment* $\mathcal{E}$ *and a malware* $\mathtt{M}$*, returns the damaged environment, namely the environment obtained by performing the malicious actions of malware* $\mathtt{M}$ *on environment* $\mathcal{E}$*.*
- Infect: $\mathtt{M}$ *replicates itself in* $\mathcal{E}$*:* $\mathtt{M}, \mathcal{E} \overset{\mathbb{R}_{\mathtt{M}}}{\Longrightarrow} \mathtt{M}', \mathtt{Rep}(\mathcal{E}; \mathtt{M})$*, where* $\mathtt{Rep} : \wp(\mathit{Progr}) \times \mathit{Progr} \rightarrow \wp(\mathit{Progr})$ *is the replication function that, given an environment* $\mathcal{E}$ *and a malware* $\mathtt{M}$*, returns the infected environment, namely the environment obtained by infecting a program of environment* $\mathcal{E}$ *with malware* $\mathtt{M}$*.*

Let us show how by further specifying functions $\mathtt{Pl}$ and $\mathtt{Rep}$ we can instantiate the above general definition to the different kind of existing malware. Given a malicious code $\mathtt{V} = (\Sigma_{\mathtt{V}}, \mathbb{R}_{\mathtt{V}})$ and a host program $\mathtt{Q} = (\Sigma_{\mathtt{Q}}, \mathbb{R}_{\mathtt{Q}})$, sometimes we write $h(\mathtt{V}, \mathtt{Q}) = (\Sigma_{\mathtt{Q}}^{\mathtt{V}}, \mathbb{R}_{\mathtt{Q}}^{\mathtt{V}})$ where $\Sigma_{\mathtt{Q}}^{\mathtt{V}}$ denotes the system obtained by infecting the system $\Sigma_{\mathtt{Q}}$ with system $\Sigma_{\mathtt{V}}$ and $\mathbb{R}_{\mathtt{Q}}^{\mathtt{V}}$ denotes the rules obtained by infecting the rules of $\mathbb{R}_{\mathtt{Q}}$ with $\mathbb{R}_{\mathtt{V}}$.

Virus. A virus is a self-propagating program that attaches itself to host programs and propagates when the hosting program executes. Some viruses are designed to damage the machines on which they execute, while other viruses simply replicate themselves.

Definition 9. *Let $\mathcal{E} = \mathtt{P}_1 \dots \mathtt{P}_n$, with $\mathtt{P}_i = (\Sigma_i, \mathbb{R}_i)$ where every element $\mathtt{P}_i$ models a program. A program $\mathtt{V} = (\Sigma_V, \mathbb{R}_V)$ is a* virus w.r.t. *$\mathcal{E}$ if, given a hosting program $\mathtt{Q} = (\Sigma_Q, \mathbb{R}_Q)$, we have that $h(\mathtt{V}, \mathtt{Q}) = (\Sigma_Q^V, \mathbb{R}_Q^V)$ is a malware w.r.t. $\mathcal{E}$ such that:*

- *Imitate:* $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_Q}{\Longrightarrow} \mathtt{Im}(\mathcal{E}; \mathtt{Q})$.
- *Injure (optional):* $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_Q^V}{\Longrightarrow} h(\mathtt{V}, \mathtt{Q})', \mathtt{Pl}(\mathcal{E}; \mathtt{V})$. *Function $\mathtt{Pl}(\mathcal{E}; \mathtt{V})$ does not depend on $\mathtt{Q}$, and this models the fact that the payload of a virus does not depend on the program hosting it.*
- *Infect:* $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_Q^V}{\Longrightarrow} h(\mathtt{V}', \mathtt{Q}), \mathcal{E}[\mathtt{P}_j \leftarrow h(\mathtt{V}, \mathtt{P}_j)]$.

Given the analogy between the proposed definition and the one of Adleman it is not surprising that we are able to prove that our notion of virus is at least as expressive as the one of Adleman. In particular, since κ is Turing equivalent [13] we show that for every recursive function that is a virus according to Adleman there exists a program in κ that computes it and that is a virus w.r.t. every environment.

Theorem 1. *Every program $\mathtt{V} = (\Sigma_V, \mathbb{R}_V)$ in the κ-framework that computes a function T_v : Progr $\rightarrow$ Progr that is a virus according to Adleman is a virus with respect to every environment according to Definition 9.*

As done by Zuo and Zhou, Adleman's definition could be instantiated to different kinds of viruses by specifying the infection strategy [26]. The same approach can be applied to our model by further delineating the infection function h.

Trojan horse, Spyware, Trap Door and Logic Bomb. Here we group together those malware that do not exhibit an infection behavior. A *Trojan horse* is a non-replicating program that hides its malicious intent inside host programs that may look useful, or at least harmless. *Spyware* are malicious programs designed to monitor user's actions in order to collect private information and send them to an external entity over the Internet. A *trap door* is a malicious code that provides a secret entry point into a program that allows someone that is aware of the trap door to gain access without going through the usual security access procedure. A *logic bomb* is a malicious code embedded in a legitimate program that executes when a certain predefined event occurs.

Definition 10. $\mathtt{V} = (\Sigma_V, \mathbb{R}_V)$ *is a* Trojan horse *or a* spyware *or a* trap door *or a* logic bomb w.r.t. *$\mathcal{E}$ if $\mathtt{M} = h(\mathtt{V}, \mathtt{Q}) = (\Sigma_Q^V, \mathbb{R}_Q^V)$ is a malware w.r.t. to $\mathcal{E}$ such that:*

- *Imitate:* $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_Q}{\Longrightarrow} \mathtt{Im}(\mathcal{E}; \mathtt{Q})$.
- *Injure:* $h(\mathtt{V}, \mathtt{Q}), \mathcal{E} \overset{\mathbb{R}_Q^V}{\Longrightarrow} h(\mathtt{V}', \mathtt{Q}), \mathtt{Pl}(\mathcal{E}; \mathtt{Q})$, *where Trojan horses, spyware, trap doors and logic bombs are differentiate by the specification of function* $\mathtt{Pl}(\mathcal{E}; \mathtt{Q})$.

k-ary virus. Interestingly, the theoretical model of k-ary virus can be seen as an instance of our definition of malware. A k-ary virus is a family of k files (some of them may not be executable) whose union constitutes a computer virus [14]. We can model the k fragments composing the k-ary virus as a set of k systems: $\{\mathtt{V}_1, \dots, \mathtt{V}_k\} = \{(\Sigma_{\mathtt{V}_1}, \mathbb{R}_{\mathtt{V}_1}), \dots, (\Sigma_{\mathtt{V}_k}, \mathbb{R}_{\mathtt{V}_k})\}$.

Definition 11. *Let $\mathcal{E} = P_1, \dots, P_n$ where each P_i models a program. We say that programs $\{\mathtt{V}_1, \dots, \mathtt{V}_k\}$ form a k-*ary virus w.r.t.* $\mathcal{E}$ if $h(\mathtt{V}_1, \mathtt{Q}_1) \dots h(\mathtt{V}_k, \mathtt{Q}_k)$ where $\mathbb{R}_V = \mathbb{R}_{V_1} \cup \dots \cup \mathbb{R}_{V_k}$ behave as follows:*

- *Imitate:* $h(\mathtt{V}_1, \mathtt{Q}_1) \dots h(\mathtt{V}_k, \mathtt{Q}_k), \mathcal{E} \overset{\mathbb{R}_{Q_l}}{\Longrightarrow} \mathtt{Im}(\mathcal{E}; \mathtt{Q}_l)$*, for every* $l \in [1, k]$.
- *Injure:* $h(\mathtt{V}_1, \mathtt{Q}_1) \dots h(\mathtt{V}_k, \mathtt{Q}_k), \mathcal{E} \overset{\mathbb{R}_{V}}{\Longrightarrow} h(\mathtt{V}_1, \mathtt{Q}_1)' \dots h(\mathtt{V}_k, \mathtt{Q}_k)', \mathtt{Pl}(\mathcal{E}; \mathtt{V}_1 \dots \mathtt{V}_k)$
- *Infect:* $h(\mathtt{V}_1, \mathtt{Q}_1) \dots h(\mathtt{V}_k, \mathtt{Q}_k), \mathcal{E} \overset{\mathbb{R}_{V}}{\Longrightarrow} h(\mathtt{V}_1, \mathtt{Q}_1)' \dots h(\mathtt{V}_k, \mathtt{Q}_k)', \mathcal{E}[\mathtt{P}_j \leftarrow h(\mathtt{V}_1, \mathtt{P}_j) \dots \mathtt{P}_{j+k} \leftarrow h(\mathtt{V}_k, \mathtt{P}_{j+k})]$.

Worm. A worm is a malicious program that uses a network to send copies of itself to other systems and, unlike viruses, do not need a host program. In general, worms do not contain a specific payload but they are only designed to spread. In order to model worms in the κ-framework we have to observe their behavior in a network. Thus, we consider an environment $\mathcal{E} = \mathtt{S}_1 \dots \mathtt{S}_n$ where each system $\mathtt{S}_i = (\Delta_i, \mathbb{R}_i)$ with $i \in [1, n]$, models a machine in a network. In this setting, when a machine $\mathtt{S}_k = (\Delta_k, \mathbb{R}_k)$ hosts a program $\mathtt{W} = (\Sigma_{\mathtt{W}}, \mathbb{R}_{\mathtt{W}})$ we simply write $\mathtt{S}_k^{\mathtt{W}} = \mathtt{S}_k, \mathtt{W}$.

Definition 12. *Let $\mathcal{E} = \mathtt{S}_i \dots \mathtt{S}_n$ where each $\mathtt{S}_i = (\Delta_i, \mathbb{R}_i)$ is a machine. A program $\mathtt{W} = (\Sigma_W, \mathbb{R}_W)$ is a* worm w.r.t. *$\mathcal{E}$ if the machine $\mathtt{S}_k^{W} = (\Delta_k, \mathbb{R}_k), (\Sigma_W, \mathbb{R}_W)$ is a malware w.r.t. $\mathcal{E}$ with the following behavior:*

- *Injure (optional):* $(\Sigma_W, \mathbb{R}_W), (\Delta_k, \mathbb{R}_k), \mathcal{E} \overset{\mathbb{R}_W}{\Longrightarrow} (\Sigma'_W, \mathbb{R}_W), (\Delta'_k, \mathbb{R}_k), \mathtt{Pl}(\mathcal{E}; \mathtt{W})$.
- *Infect:* $(\Sigma_W, \mathbb{R}_W), (\Delta_k, \mathbb{R}_k), \mathcal{E} \overset{\mathbb{R}_W}{\Longrightarrow} (\Sigma'_W, \mathbb{R}_W), (\Delta_k, \mathbb{R}_k), \mathcal{E}[\mathtt{S}_i \leftarrow \mathtt{S}_i, (\Sigma_W, \mathbb{R}_W)]$.

Botnet. The term *botnet* refers to a network of compromised machines. A botnet is controlled by the so called botmaster, that is an attacker machine that performs the following actions: (1) identifies a set of target machines; (2) installs a bot on each target machine such that the bot remains dormant until it receives a predefined command form the botmaster; (3) launches the attack by triggering the bots. Botnet are successfully used to implement distributed denial of service attacks.

Definition 13. *A program $\mathtt{B} = (\Sigma_B, \mathbb{R}_B)$ is a* botmaster w.r.t. *$\mathcal{E} = \mathtt{S}_1 \dots \mathtt{S}_n$, where each $\mathtt{S}_i$ is a machine in a network, if it is a malware w.r.t. $\mathcal{E}$ that either infects: $\mathtt{B}, \mathcal{E} \overset{\mathbb{R}_B}{\Longrightarrow} \mathtt{B}', \mathcal{E}[\mathtt{S}_i \leftarrow h(\mathtt{S}_i, \mathtt{bot})]$, or injures: $\mathtt{B}, \mathcal{E} \overset{\mathbb{R}_B}{\Longrightarrow} \mathtt{B}', \mathtt{Pl}(\mathcal{E}; \mathtt{bot}, \mathtt{B})$. A program $\mathtt{bot}$ is a bot if every machine $h(\mathtt{S}_i, \mathtt{bot})$ hosting it is a malware w.r.t. $\mathcal{E}$ that behaves as follows:*

- *Imitate:* $\mathtt{B}, \mathcal{E}[\mathtt{S}_i \leftarrow h(\mathtt{S}_i, \mathtt{bot})] \overset{\mathbb{R}_{S_i}}{\Longrightarrow} \mathtt{Im}(\mathtt{B}, \mathcal{E}[\mathtt{S}_i \leftarrow h(\mathtt{S}_i, \mathtt{bot})]; \mathtt{S}_i)$.
- *Injure:* $\mathtt{B}, \mathcal{E}[\mathtt{S}_i \leftarrow h(\mathtt{S}_i, \mathtt{bot})] \overset{\mathbb{R}_{bot}}{\Longrightarrow} \mathtt{Pl}(\mathtt{B}, \mathcal{E}[\mathtt{S}_i \leftarrow h(\mathtt{S}_i, \mathtt{bot})]; \mathtt{B})$.

Thus, the proposed definition of malware in κ allows for the specification of the main actors involved in a botnet attack. In particular, thanks to the different levels of granularity that we can use to model a κ-system, we are able to specify both the botnet as a whole and the single bots in the same framework, and we can tune the level of abstraction of these specifications according to the features that we want to analyze. For instance, since bots are given by the combination of standard malware, like viruses and worms, and a communication channel with the botmaster, we can precisely model in κ both the behavior of the bots and their interaction with the botmaster.

4 Hunting Malware in κ

The typical approach to malware defense is malware detection, namely the identification of the malicious code in order to avoid infection. In literature there are many results that show the intractability of the malware detection problem (e.g. [1,4,7,17,26]). All these results share a common pattern: (i) they provide a formalization of malware in a Turing complete model of computation (e.g., Turing machines, recursive functions); (ii) they prove the undecidability of the malware detection problem by reducing it to problems that are known to be undecidable. In Section 3 we have proposed a formalization of the notion of malware in κ, a Turing complete calculus [13]. In particular, according to our definition of malware, in order to decide whether a program $\mathtt{M}$ is a malware w.r.t. an environment $\mathcal{E}$ we have to verify that every evolution of $\mathcal{E}$ is also an evolution of $\mathcal{E},\mathtt{M}$ (imitate); that $\mathcal{E},\mathtt{M}$ allows more evolutions than $\mathcal{E}$ (injure); and that $\mathcal{E},\mathtt{M}$ has evolutions that lead to the replication of $\mathtt{M}$, namely $\mathcal{E},\mathtt{M} \Rightarrow \mathcal{E}',\mathtt{M},\mathtt{M}'$ with $\mathtt{M}\doteqdot\mathtt{M}'$ (infect). Observe that the problem of deciding whether the evolutions of $\mathcal{E}$ are included in the ones of $\mathcal{E},\mathtt{M}$ can be reduced to the problem of deciding program equivalence which is known to be undecidable in any Turing complete language. Moreover, the problem of malware replication can be reduced to a coverability problem which is known to be undecidable in κ [13]. Thus, as expected, also with our formalization of malware it is possible to conclude that malware detection is undecidable.

A more recent promising approach to malware defense regards the monitoring of malware propagation in a network both for studying properties of the considered malware and malware epidemiology and for finding ways to immunize the network.

Malware Propagation. The problem we want to tackle can be made clear with a similitude with the biological setting. Suppose that a certain infection is taking off and degenerating into an epidemics: What is the faster way to vaccinate people in order to minimize the spread of the infection? Rephrasing this problem in our setting: suppose that a network is under the attack of a fast spreading malware, how can we detect which are the weak nodes to be immunized? This is not a new approach to the field, several studies have been conducted to this aim. The first attempt dates back to 1991 when Kephart and White [18] propose an epidemiological model inspired by the biological setting. More recently, this work has been shown to be outdated as modern worms spread with different kinds of models usually guided by the topology of the network [15,19,23]. A key aspect of these techniques is that they employ stochastic measures to represent both the speed of propagation and the likelihood of a node to be infected (stochastic measures could also model network features such as link capacities, congestion, intermittent connectivity). Interestingly, as mentioned in the introduction, there exists simulation tools for κ-systems that take into account stochastic measurements (expressing the probability of a rule to be applied) [8]. Let us detail the main steps for the investigation of malware propagation by means of κ and its simulation tools:

1. *Modeling:* Choose the proper level of abstraction of the model and formalize the main actors as κ-systems and their activities as sets of rules. Then, set the stochastic metrics either as the result of personal investigation or derived from previous ex-

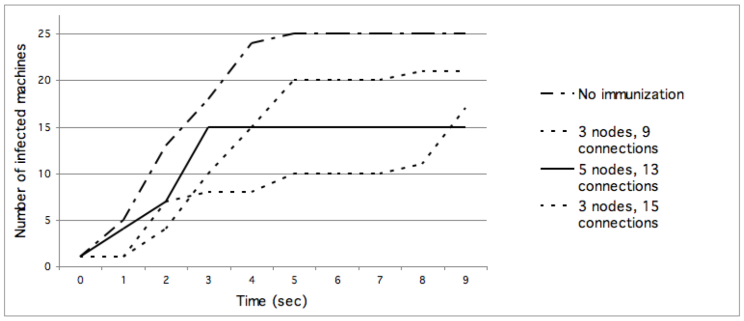


Fig. 2. Speed of infection of Example 1

isting works (e.g. [10,15]). This leads to the specification of a stochastic κ-system that represents the initial condition of the network.

2. *Simulation:* Choose one of the freely available tools in [8] that, given the specification of the model in κ, simulates its evolution and returns the resulting system.
3. *Analysis:* From the results of the simulation, we extract the *network of infected machines*. This accounts in showing which machines have been infected and which connections have been exploited for malware propagation (the topology of this network could be substantially different from the original one). By analyzing this graph it should be possible to determine the best defense strategy to obstruct the propagation of the considered malware. For instance, the machines to be immunized could be identified by running a discovery algorithm on the graph that counts the nodes reachable, and therefore corruptible, from each node.

Example 1. We have considered and modeled in the κ-framework a small network of 25 machines and a malware infecting it. In one step the malware infects all its sane neighbors that are not immunized. We have simulated the spread of the infection when no machines are immunized and in few seconds all the network has been compromised. In this simple case, in order to identify the best nodes to be immunized on the network of infected machines it is enough to count the number of connections of each nodes. We have considered different immunization scenarios and the experimental results confirm that the speed of infection decreases with the growth of the number of connections of the immunized nodes. Fig. 2 shows the speed of infections when different sets of nodes (and therefore connections) are immunized.

The special case of botnets. As depicted in several works [9,11,25], botnets are difficult to track. Although being a major threat, their life-cycle is still yet to be understood and it is almost impossible to prevent systems from being compromised. Lately, researchers have focused on the study of botnet propagation strategies, in order to make the botnet harmless by arresting the spread of bots.

The systematic strategy proposed above could be used for the study of the recruitment phase performed by bots infection. Indeed, botnets usually exploit standard malicious techniques to corrupt and therefore recruit target machines. For this reason, we claim that by modeling botnets in the κ-framework, we could shed light on the life-cycle of botnets and on their propagation strategies. Roughly speaking, we could represent different topologies of networks where nodes model the machines and the botmaster. Machines are equipped with a special state that indicates if they are clean or infected and edges represent the physical connections between them. Rules describe both the choice of potential victims in the recruitment phase and how the network changes along time. Thus, from the analysis of the network of compromised machines obtained through the simulation process, we could extract important features about the botnet behavior.

5 Concluding Remarks

In this work we have proposed to apply biological concurrent calculi to the analysis of malicious code. We have focused on the κ-calculus and we have used it to provide an unifying formalization of existing malware. We have analyzed and discussed the potentialities of modeling malware and their propagation in the κ-framework. We believe that the use of κ will open new challenges in the field of malware analysis and defense. In particular, we claim that the proposed formalization of malware provides a powerful framework for understanding the behavior of distributed malware. In fact, we believe that κ and its tools of analysis provide the right means for investigating the main aspects of distributed malware: propagation over a network, machines and programs interactions, launch of a distributed attack. By simulating the distributed malware behavior it is possible to identify the countermeasures to take in order to defend an environment from the considered attack, or measure the goodness of existing defense strategies. As for future work in this direction, we plan to model with κ real types of botnets [2] and to use the metrics proposed in known works (e.g. [10,15]) in order to stochastically simulate their behavior. We believe that considerably big portions of the Internet network can be faithfully compiled into κ and by observing the results of the simulation, we will be able to better understand the behavior of the considered botnets and propose an efficient defense strategy.

In this work, we have focused on the distributed features of the κ-calculus to reason on network properties and spreading behaviors. However, the existence of decidability results in κ [13] opens the way for the study of interesting reachability malware properties that could be used for malware detection. This means that this investigation could lead to the identification of fragments of the κ-calculus where the detection of specific classes of malware becomes decidable.

Acknowledgements. We would like to thank Maurizio Gabbrielli and Roberto Giacobazzi for their support and helpful feedbacks. We are also grateful to Jean Krivine and Jorge A. Pérez for their remarks and suggestions.

References

1. L. M. Adleman. An abstract theory of computer viruses. In *CRYPTO*, volume 403 of *LNCS*, pages 354–374, 1988.

2. P. Bächer, T. Holz, M. Kötter, and G. Wicherski. Know your enemy: Tracking botnet. http://www.honeynet.org/papers/bots.
3. J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.
4. G. Bonfante, M. Kaczmarek, and J. Marion. On abstract computer virology from a recursion theoretic perspective. *Journal in Computer Virology*, 1(3-4):45–54, 2006.
5. L. Cardelli. Brane calculi. In *CMSB*, volume 3082 of *LNCS*, pages 257–278, 2005.
6. L. Cardelli and A. D. Gordon. Mobile ambients. *TCS*, 240(1):177–213, 2000.
7. F. Cohen. Computer viruses: Theory and experiments. *Computers and Security*, 6:22–35, 1987.
8. Collection of kappa tools. http://kappalanguage.org/tools.
9. E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *SRUTI 2005*, pages 39–44, 2005.
10. D. Dagon, G. Gu, and C. P. Lee. A taxonomy of botnet structures. In *Botnet Detection*, volume 36 of *Advances in Information Security*, pages 143–164. Springer, 2008.
11. D. Dagon, C. C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *NDSS*. The Internet Society, 2006.
12. V. Danos and C. Laneve. Formal molecular biology. *TCS*, 325(1):69–110, 2004.
13. G. Delzanno, C. Di Giusto, M. Gabbrielli, C. Laneve, and G. Zavattaro. The kappa-lattice: Decidability boundaries for qualitative analysis in biological languages. In *CMSB 2009*, volume 5688 of *LNBI*, 2009.
14. E. Filiol. Formalisation and implementation aspects of k-ary (malicious) codes. *Journal in Computer Virology*, 3(2):75–86, 2007.
15. A. J. Ganesh, L. Massoulié, and D. F. Towsley. The effect of network topology on the spread of epidemics. In *INFOCOM*, pages 1455–1466. IEEE, 2005.
16. R. A. Grimes. Malicious mobile code: Virus protection for windows. *O'Reilly & Associates, Inc.*, 2001.
17. G. Jacob, E. Filiol, and H. Debar. Formalization of viruses and malware through process algebras. In *ARES'10*, pages 597–602. IEEE Computer Society, 2010.
18. J. O. Kephart and S. R. White. Directed-graph epidemiological models of computer viruses. *Security and Privacy, IEEE Symposium on*, 0:343–361, 1991.
19. J. Kim, S. Radhakrishnan, and S. K. Dhall. Measurement and analysis of worm propagation on internet network topology. In *ICCCN*, pages 495–500. IEEE, 2004.
20. B. McCarty. Botnets: Big and bigger. *IEEE Security and Privacy*, 1:87–90, 2003.
21. G. McGraw and G. Morrisett. Attacking malicious code: Report to the Infosec resarch council. *IEEE Software*, 17(5):33–41, 2000.
22. R. Milner. *Communication and concurrency*. Prentice Hall International, 1989.
23. B. A. Prakash, H. Tong, N. Valler, M. Faloutsos, and C. Faloutsos. Virus propagation on time-varying networks: theory and immunization algorithms. In *ECML PKDD'10*, pages 99–114. Springer-Verlag, 2010.
24. D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
25. Q. Wang, Z. Chen, C. Chen, and N. Pissinou. On the robustness of the botnet topology formed by worm infection. In *GLOBECOM*, pages 1–6. IEEE, 2010.
26. Z. Zuo and M. Zhou. Some further theoretical results about computer viruses. *Computer Journal*, 47(6):627–633, 2004.