J Comput Virol (2007) 3:75–86
DOI 10.1007/s11416-007-0044-2

EICAR 2007 BEST ACADEMIC PAPERS

Formalisation and implementation aspects of K-ary (malicious) codes

Eric Filiol

Received: 12 January 2007 / Revised: 9 February 2007 / Accepted: 15 March 2007 / Published online: 22 May 2007

Abstract This paper presents a new class of (malicious) codes denoted k-ary codes. Instead of containing the whole instructions composing the program's action, this type of codes is composed of k distinct parts which constitute a partition of the entire code. Each of these parts contains only a subset of the instructions. When considered alone (e.g. by an antivirus) every part cannot be distinguished from a normal uninfected program while their respective action combined according to different possible modes results in the offensive behaviour. In this paper, we presents a formalisation of this type of codes by means of Boolean functions and give their detailed taxonomy. We first show that classical malware are just a particular instance of this general model then we specifically address the case of k-ary codes. We give some complexity results about their detection based on the interaction between the different parts. As a general result, the detection is proved to be NP-complete.

Keywords Cohen model · k-ary malware · Detection problem · Polymorphism · Metamorphism · Code interaction

1 Introduction

In the Cohen's general model of computer viruses [2], every code is made of a single program which contains the whole instructions devoted to its action. Let us consider his now famous definition of a virus:

E. Filiol (✉)
Ecole Supérieure et d'Application des Transmissions,
Laboratoire de virologie et de cryptologie, B.P. 18,
35998 Rennes Armées, France
e-mail: eric.filiol@esat.terre.defense.gouv.fr

Definition 1 A *virus* can be described by a sequence of symbols which is able, when interpreted in a suitable environment (a machine), to modify other sequences of symbols in that environment by including a, possibly evolved, copy of itself.

Since every virus is supposed to be composed of a single code, antiviral detection itself considers only this model. As a consequence, it is eventually possible to decide (up to a complexity or decidability issue) whether the program is infected or not. All the information with respect to the final offensive action is contained in a single program.

But recently, some new yet trivial attempts (*Scob/Paddodor*, *Perrun*...) in malware writing seems to announce a new type of malware made of set of files acting in a combined manner. Despite the fact that the few known examples are rather very trivial, the objective is to scatter the viral information over different files and thus make the viral detection far more complex: each of the k constituting part looks like an innocent file and thus does not trigger any alert. But beyond these trivial attempts, there is a tremendous risk that in a very near future, antivirus software would be defeated by sophisticated variants of such codes.

We denote this new type of codes *k-ary viruses*.[1] Let us give a precise definition of these codes.

Definition 2 [5,6] A *k-ary virus* is a family of k files (some of them may be not executable) whose union constitutes a computer virus and performs an offensive action that is equivalent to that of a true virus. Such a code is said *sequential* (*serial mode*) if the k constituent parts are acting strictly one after the another. It is said *parallel* if the k parts executes simultaneously (parallel mode).

[1] The concept of k-ary viruses can be extended to any other type of malware: self-reproducing or simple programs. For sake of simplicity and without loss of generality, we will limit ourselves to the computer viruses.

It is worth mentioning that the two modes can be combined as the *sequential/parallel mode*. Each of these working modes may be efficiently considered according to their respective properties:

- With sequential mode, it is possible to use non-executable file format in order to scatter the viral information (image file, sound file, encrypted data...). This mode is used to perform *process escape* or *process hopping*.
- With parallel mode, a combinatorial effect works against the antivirus' action. The latter has to pick up k processes among all the running processes in a machine. With an efficient implementation of this type of code, traditional detection has thus exponential complexity. This mode is used to perform *process sprawl*.

In this paper, we present a new model enabling to precisely describe and study k-ary viruses. We identify the different classes of k-ary viruses. In particular, we focus on the problem of their detection and show that in the general case, this problem is NP-complete. This study has been initiated by existing yet unknown malware and it has been later validated by proof-of-concept programs in our laboratory. Whatever the system analysis tools or security software we may use, these codes remain undetected. This results shed a rather pessimistic light on the future of computer attacks.

The paper is organised as follows. In the first section, we present the new formalisation tools and framework we use to study k-ary viruses. Essentially, we consider vector Boolean functions. Then we compare our model to the Cohen's classical model [2]. In particular, we show that our model is a generalisation of Cohen's one, despite the fact that our approach is totally different. Simple viruses and polymorphic/metamorphic viruses are presented as particular instances of our model. In the next section, we then extensively present k-ary malware in sequential mode. The POC_SERIAL worm is taken as an illustrative example. The last but one section is considering k-ary malware in parallel mode. The PARALLEL_4_S virus will illustrate the theoretical results established for that family of k-ary codes. Finally, in the conclusion, we address future work and some open problems.

2 The new formalisation framework

This framework is based on vector Boolean functions instead of Turing machines as in the Cohen's model [2]. The essential reason for that choice comes from the fact that Turing machines cannot thoroughly describe the interaction between programs, even by considering k-Turing machines (multitapes machines). Besides the fact that it would far too complex to consider them as formalisation tools, it has been shown [6] that the generalisation of the Cohen's model is too limited.

In order to efficiently model k-ary malware, the concept of vector Boolean functions is far more powerful. It enables to consider simultaneously different files whatever may be their respective status (executable or not). In order to define the working context, let is consider an operating system containing n files (n is of arbitrary size). These are all possible files that exist or may exist in the system at a given time. Each of these files are described by a Boolean variable x_i, $i = 1, 2, \ldots, n$. No particular assumptions is made about the status of any of these files (executable or not, data...).

We set $x_i = 1$ whenever the file i is considered, otherwise $x_i = 0$ (in particular, if it does not exist at time instant t). Instead of working with a variable number of files, it is more efficient to fix n arbitrary large. From a mathematical point of view, whenever given files $i_{j_1}, i_{j_2}, \ldots, j_{i_m}$ do not exist at time instant t, then any Boolean function f_t over $\mathbb{F}_2^n$ we may use, will be degenerated in the corresponding variables, where $\mathbb{F}_2$ denotes the binary field.

We then consider two different Boolean functions: the *transition function* and the *infection function*. Both describe the relationship which exists between the files in the system at time instant t.

2.1 Preliminary concepts

Most of self-reproducing codes that exist at the present time are worms and thus a single copy of the malware is present in the system. But it is not the case as far as virus are considered (many copies exist in the system at the same time). In order for our model to be general, we will consider that all the different copies of a malicious codes are in fact a single one, e.g. the viral code. In the very special case of k-ary viruses, the viral code is the disjoint union of k different files.

From a mathematical point of view, it is equivalent to consider the following equivalence relation, which is defined on a set S (the file system).

Definition 3 (Infection relation) Let x_1 and x_2 be two files and let v be a given computer virus. We then define the equivalence relation $\mathcal{R}_v$ as follows:

$x_1 \mathcal{R}_v x_2$ if $x_1 \cap x_2 \in \{x_1, x_2, v\}$.

This is an equivalence relation and any equivalence class of a given element x is defined by $C(x) = \{y | y \in S \text{ and } x \mathcal{R}_v y\}$. The class $C(v)$ contains every file infected by v. Every class which is a singleton contains an uninfected file.

Consequently, in the rest of this paper we will consider the quotient set with respect to $\mathcal{R}_v$. It is a partition of the set S and we denote it $S_{/\mathcal{R}_v}$. Thus once a file has been infected, it will be considered as equivalent to the virus v itself. This assumption is natural since an infected file spreads itself the infection and hence behave like v. From now on, the use of the quotient set will be implicitly considered.

2.2 The transition function

It is a vector Boolean function, denoted $F^t : \mathbb{F}_2^n \to \mathbb{F}_2^n$. It describes the relationship and the interactions existing between files from time instant $t-1$ to time instant t. In particular, as far as a k-ary code is concerned (suppose its parts are files $i_{j_1}, i_{j_2}, \ldots, j_{i_k}$), the transition function will describe the interactions between its k different, constituent parts from one side and its interaction with the other files in the system in the other side.

Without loss of generality and for clarity's sake, it is more convenient to consider the functional restriction of F^t to F_v^t, the latter considering the k files of the virus only. This restriction enables to focus on their relationship exclusively.

There exist different representation for the transition function F^t. The first one is its truth table as in Table 1. It represents the *system global transition*.

The coordinate functions F_1^t, F_2^t, F_3^t of F^t are the functions over $\mathbb{F}_2$ such that $F^t = (F_3^t, F_2^t, F_1^t)$. It is then possible to compute their Disjunctive Normal Form (DNF) by means of the Möbius transform [4]. Thus the function described in Table 1 yields:

$$F_1^t(x_3, x_2, x_1) = x_2 x_3$$
$$F_2^t(x_3, x_2, x_1) = x_1 \overline{x_3}$$
$$F_1^t(x_3, x_2, x_1) = \overline{x_2}$$

This representation enables us to see that file 1 interacts only with file 3. More generally, $F_i^t(x_3, x_2, x_1)$ depends on files interacting with file i only. Finally, we can equivalently and more compactly use the associate incidence matrix $I(F^t)$ to describe the transition function. For the function of Table 1, we have:

$$I(F^t) = \begin{vmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{vmatrix}$$

More generally, the incidence matrix $I(F^t)$ is a square matrix of order n which is defined as follows:

Table 1 A truth table for a vector Boolean function ($n = 3$)

x_3	x_2	x_1	$F_3^t(x_3, x_2, x_1)$	$F_2^t(x_3, x_2, x_1)$	$F_1^t(x_3, x_2, x_1)$
0	0	0	0	0	1
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	1	0	0

$a_{ij} = 1$ if file i acts on file j,

$a_{ij} = 0$ otherwise.

We can graphically describe the information contained in the incidence matrix $I(F^t)$ by means of its associate *transition graph* as depicted in Fig. 1.

The transition graph—or equivalently its incidence matrix—describes file interactions at a rather global level only (all-or-nothing information). We only know whether a file acts on another file or not but not how it does. That is the reason why we must consider the *iteration graph* in order to have a more detailed information on the infection process itself. It is established by finitely iterating the transition function F^t with respect to time t as follows:

$$\begin{cases} x^0 \in \mathbb{F}_2^n \\ x^{t+1} = F^t(x^t) \quad t = 0, 1, 2, \ldots \end{cases}$$

The resulting iteration graph is then depicted in Fig. 2, each node being the decimal value of triplets (x_3, x_2, x_1) computed as $4x_3 + 2x_2 + x_1$. The iteration graph describes the dynamic evolution of the system S at each time instant t with respect to the infection by the different parts composing the k-ary virus v. In particular, it is worth mentioning the existence of a fixed point (or stable state for the system S) which is obtained after four infection steps (all possible target files have been infected).

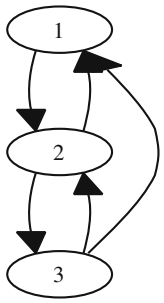


Fig. 1 Transition graph of Table 1

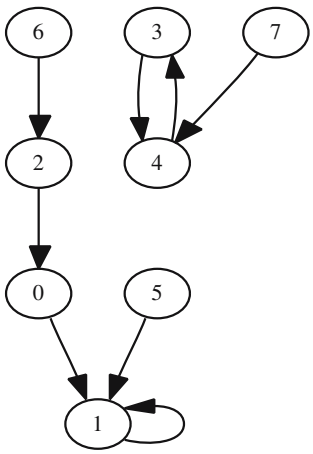


Fig. 2 Iteration graph of Table 1

2.3 The infection function

The *infection function* is used to describe the different parts which compose the virus. Somehow this function can also be seen as the detection function since it shows whether the virus is present or not in the system S or equivalently, it is possible to state the presence of the virus in S. It enables to describe how the viral information is scattered all over this system.

The infection function is consequently defined by $f_v : \mathbb{F}_2^n \to \mathbb{F}_2^n$. For any $x \in \mathbb{F}_2^n$, the system S is infected by the virus v if and only if $f_v(x) = 1$. The interest of the infection function may look somehow artificial, since most of the time it is unknown. It is a *a posteriori* knowledge. But this function is very useful for our formalisation as we will see later.

Composing infection and transition functions From a practical point of view, it is worth considering both the system evolution at a given time instant t and its infectious status (infected or not) at the same time. While for classical computer viruses ($k = 1$) both are in fact equivalent, it is not the case with k-ary viruses. Despite the presence in the system S of a k'-subset of the k parts of a k-ary virus ($k' < k$), it does not necessarily imply that S is infected. A trivial example is that of source code viruses. The viral source code will not trigger any alert while the pair code/compiler should. Companion viruses could be considered as another yet trivial example of k-ary viruses.

From a mathematical point of view, we will consider the functional composition of function F^t with function f_v, denoted $f_v \circ F^t$. Thus, if there exists $x \in \mathbb{F}_2^n$ such that $(f_v \circ F^t)(x) = f_v(F^t(x)) = 1$, we will state that x has infected the system S at time instant t.

Before presenting the main results we have obtained from our model, let us give the working framework we use for sake of simplicity and clarity at least in a first formalisation step.

Convention 1 *In any real-life system S, the different interactions existing between files are very numerous and complex: file creation or deletion, call, links... It would have been impossible to describe all of these due to the file system complexity. Without loss of generality, we will not model these interactions unless strictly necessary. The corresponding sub-model—in other words the model without considering the interactions with non-viral files—will be denoted "*simplified model*". However whenever suitable for some particular results, these interactions will be referred as the "*generalised model*".*

3 Classical malware in the new model

Despite the fact that our model is clearly different from Cohen's one, it encompasses the latter. In order to illustrate our claim, we will consider to main cases: simple viruses and polymorphic viruses—or equivalently, according to Cohen's model [2], singleton viral set and largest viral set respectively. Each of them is just one program—possibly an evolved form of another one—which performs a complete viral action on its own. Consequently, these "classical" viruses are k-ary viruses indeed with $k = 1$.

3.1 Case of a simple virus (Cohen's singleton viral set)

Let us consider a toy system with three files x_1, x_2 et x_3. The virus v will be described by file x_1, while file x_2 will be a target executable file with respect to v and file x_3 will not (e.g. data file). Let us recall (see the section devoted to preliminary concepts) that we work up to the equivalence relation $\mathcal{R}_v$ by considering the following functional composition:

$$S \xrightarrow{F^t} S \xrightarrow{\mathcal{R}_v} S_{/\mathcal{R}_v}.$$

The infection of file x_2 by the file x_1 is then described by the sequence

$$(0, 1, 1) \overset{F^t}{\longmapsto} (0, 1, 1) \overset{\mathcal{R}_v}{\longmapsto} (0, 0, 1).$$

The transition function (resp. infection) F^t (resp. f_{x_1}) are given in Table 2.

Interesting combinatorial results have been stated from the study of the corresponding iteration graph depicted in

Table 2 Transition and infection functions of a singleton viral set

x_3	x_2	x_1	$F_3^t(x_3, x_2, x_1)$	$F_2^t(x_3, x_2, x_1)$	$F_1^t(x_3, x_2, x_1)$	$f_{x_1}(x_3, x_2, x_1)$
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	0	1	0	0
0	1	1	0	0	1	1
1	0	0	1	0	0	0
1	0	1	1	0	1	1
1	1	0	1	1	0	0
1	1	1	1	0	1	1

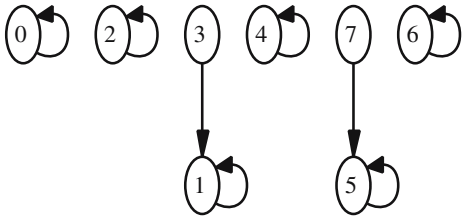


Fig. 3 Iteration graph of a singleton viral set

Fig. 3. The study of the iteration graph enables to state this first result.

Proposition 1 *Let v be a simple virus, described by the tuple (point) e_i ($x_i = 1$ and $x_j = 0$ for $j \neq i$), and let us consider S a system containing n fichiers among which m are infectible with respect to the virus v. Then, the iteration graph describing the infection with respect to v contains:*

- *2^{n-1} connected components, each of them being reduced to a single fixed point;*
- *a connected component, called viral component, which contains the fixed point e_i and $2^m - 1$ other points which all have the point e_i as a successor;*
- *2^{n-m-1} connected components which all contain a single fixed point.*

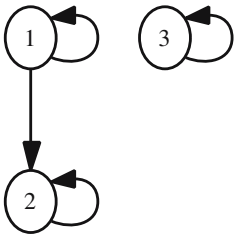
Proof Let be $x \in \mathbb{F}_2^n$. Let us denote by $\mathrm{supp}(x)$ the set $\{i \leq n | x_i = 1\}$. Let us suppose that $e_j = v$. Then three different cases are to be considered:

- if $j \notin \mathrm{supp}(x)$, we then have $F^t(x) = x$. By construction, there are exactly 2^{n-1} such points x;
- if $j \in \mathrm{supp}(x)$ and the indices k such that $k \in \mathrm{supp}(x)$ relates to a file that can be infected by v only, then we have $F^t(x) = e_j$. By construction, there are exactly $2^m - 1$ such n-tuples;
- the other cases are those for which we have $j \in \mathrm{supp}(x)$ and the indices k such that $k \in \mathrm{supp}(x)$ with x_k describing any file (that can be infected or not by the virus v) different from v. Let be two different such n-tuples x and y. We then have either $F^{t+1}(x) = y$ if $y \prec x$ (that is to say $\mathrm{supp}(y) \subsetneq \mathrm{supp}(x)$) or $F^{t+1}(x) = x$; in this latter case, it is a minimal elemeny for the ordering $\prec$. There are 2^{n-m-1} such minimal elements for that third case. Those minimal elements are all the points x whose support contains j and any subset of indices k which correspond to files that cannot be infected by v, only. Since a connected component can contain only a single fixed point only [12, pp. 20, Proposition 1.1], we have proven the result.

By considering the incidence matrix $I(F^t)$ given by

$$I(F^t) = \begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{vmatrix}$$

Fig. 4 Transition graph of a singleton viral set



we can establish the corresponding transition graph given in Fig. 4.

Then, the following proposition can be stated from the analysis of both graphs.

Proposition 2 *Let v be a singleton viral set described by the n-tuple e_i ($x_i = 1$ and $x_j = 0$ for $j \neq i$) and let S a (n, m)-file system where m is the number of files that are insensitive with respect to infection by v. Then, the incidence matrix describing the infection with respect to virus v has:*

- *$m + 1$ rows of weight 1 (a single non-null entry);*
- *$n - m - 1$ rows of weight 2;*
- *$n - 1$ columns of weight 1;*
- *one column of weight $n - m$, denoted* infection column*, which is gathering the viral point e_i and every of its targets x_j.*

Proof The proof is straightforward by considering the fact that the coordinate functions F_i^t (namely the matrix rows) with respect to the files that are insensitive to infection by v, depend by construction on these files only. Consequently, there are m rows of weight exactly 1, plus the row which corresponds to the virus file itself. On the contrary, the coordinate functions with respect to the files that can be infected by v depends on both the virus and these files. There are exactly $n - m - 1$ such rows.

As for the columns, the proof is the same by considering the possible action of the file i on the coordinate function F_j^t instead of the dependence relationship. The single column that contains more than one non-null entry is that which describes the action of the virus e_i. □

Proposition 2 is very interesting since it shows that according to our model, file interaction-based detection by means of the incidence matrix alone has linear complexity $\mathcal{O}(n)$ (weight evaluation of n columns). Unfortunately, despite this linear complexity, the size of input data grows quadratically in n and therefore may be huge. As an example, is $n = 20{,}000$, the incidence matrix has a size of 0, 4 Gb. However, sampling techniques may be successful at reducing the overall complexity. Moreover, building the incidence matrix may be performed by emulation techniques (test whether file x_i acts on file x_j). While classical detection techniques will do far better in client antivirus software, incidence based detection technique may prove very efficient for initial detec-

tion (case of unknown viruses) on honeypot machines, at the antivirus publisher or Internet monitoring center's level.

Let us now consider a statistical description of the infection which is useful for sampling based-detection. This approach is very interesting since building the transition function F^t is untractable in practice. Let us note $\text{supp}(u) = \{i | u_i \neq 0\}$ with $u = (u_1, u_2, \ldots, u_n)$.

Proposition 3 *Let $F^t(x)$ the transition function of a virus denoted by the n-tuple e_i. Let u and w be two elements of $\mathbb{F}_2^n$. Let us additionally denote the infectible files with respect to e_i by c_j and those are are not, by n_k. Then, for a singleton viral set, the probability $P[< F^t(x), u >=< x, w >]$ equals:*

- $P[< F^t(x), u >=< x, w >] = 1$ *iff* $u = w = e_i$;
- $P[< F^t(x), u >=< x, w >] = 1$ *iff* $supp(u) = supp(w)$ *and iff their support set contains the virus index i and indices k of non-infectible files with respect to e_i, only;*
- $P[< F^t(x), u >=< x, w >] = \frac{1}{4}$ *iff $w = e_i$ and iff $supp(u)$ contains indices j of infectible files with respect to e_i, only;*
- $P[< F^t(x), u >=< x, w >] = \frac{3}{4}$ *iff $supp(u)$ and $supp(w)$ contain the virus index i and /or the same infectible files c_j (in other words, the restriction of $supp(u)$ and $supp(w)$ to files c_j are identical);*
- $P[< F^t(x), u >=< x, w >] = \frac{1}{2}$ *in every other cases.*

where $< ., . >$ denotes the usual scalar product. Finally, we have

$$P[(f_v \circ F^t)(x) =< x, e_i >] = 1.$$

Proof The two first ones are obvious. Indeed, the virus cannot infect itself (if the virus is supposed to be efficiently designed) and as far as the files that are insensitive to the infection with respect to this virus are concerned, the inputs and the outputs of the function are the same.
For the third case, we have:

- Let be $i \in \text{supp}(x)$, then in this case we have $< F^t(x), u >= 0$. Indeed, every $< F^t(x), c_j >= 0$ (when considering the quotient set),
- Let be $i \notin \text{supp}(x)$ then $< F^t(x), u >= 0$ with a probability of $\frac{1}{2}$.

We then have

$$P[< F^t(x), u >=< x, v >] = P[i \in \text{supp}(x)].1 + P[i \notin \text{supp}(x)].\frac{1}{2} = \frac{1}{4}.$$

The remaining cases are proved in a similar way.

Finally, the last probability $P[(f_v \circ F^t)(x) =< x, e_i >]$ is obvious to compute in the special case of the singleton viruses. □

Proposition 3 is very interesting since it shows that input/output correlations for the transition function F^t are independent from the system S. Thus, we can characterise a singleton viral set uniquely and independently from any system S.

3.2 Case of a polymorphic/metamorphic Virus (Cohen's largest viral set)

We will assume that the viral set (LVS) we consider is finite (but we can generalise to finite enumerable sets [15]. In that set, every element is an evolved form of another element (at least by transitive closure). Thus, for a given Turing machine, we have $\forall \{v_i, v_j\} \subset LVS(M)$ $v_i \to v_j$ or $v_j \to v_i$. If the set is finite we consider that both relations hold.

The representation of polymorphic viruses by means of our Boolean model becomes untractable very soon. So, due to lack of space, we will not give *in extenso* the different possible transition functions but the different rules to build them only.

We consider a n-form polymorphic virus, denoted $(x_{v_1}, \ldots, x_{v_i}, \ldots, x_{v_n})$. Let us also consider a system S containing N files whose m are infectible with respect to the virus (hence $N-m-n$ are not). We denote $(x_{f_1}, \ldots, x_{f_j}, \ldots, x_{f_m})$ the infectible files and $(x_{i_1}, x_{i_2}, \ldots, x_{i_k}, \ldots, x_{i_{N-m-n}})$ those which are not. The general rule to build the transition function F^t are, $\forall x \in \mathbb{F}_2^n$:

- If $\text{supp}(x) \cap \{v_i\} = \emptyset$, then $F^t(x) = x$ (the virus is not active);
- If $\text{supp}(x) \cap \{v_i\} \neq \emptyset$ and $\text{supp}(x) \cap \{i_k\} \neq \emptyset$ and $\text{supp}(x) \cap \{f_j\} = \emptyset$, then $F^t(x) = x$ (the virus is active but there is no file to infect);
- Let $n_1 = |\text{supp}(x) \cap \{v_i\}|$ (the number of mutated forms represented in x) and $n_2 = |\text{supp}(x) \cap \{f_j\}|$ (the number of files that are infectible by the mutated forms in x). Then, we define $y = x$ and we cancel the $n-2$ positions in y whose index describes an infectible file and if $n_2 > n - n_1$, we set to 1 $n - n_1$ null positions in y which each corresponds to a mutated form of the virus (for every new target, a new mutated form appears, until the total number of forms has been reached). Finally, we put $F^t(x) = y$.

It is worth mentioning that any mutated form does not infect any other one (at least for a well-written code). As an example, Table 3 gives the transition and iteration functions of a 3-form polymorphic virus, denoted x_1, x_2 and x_3. The system S contains five files in which only file x_4 is infectible with respect to the virus.

The iteration graph is partially given in Fig. 5 as well as the transition graph in Fig. 6 (single fixed points have been omitted). Let us state a first result that be derived from our model.

Table 3 Transition and iteration functions of a polymorphic/metamorphic virus

$(x_5, x_4, x_3, x_2, x_1)$	$F_5^t(x)$	$F_4^t(x)$	$F_3^t(x)$	$F_2^t(x)$	$F_1^t(x)$	$f_{\{x_1,x_2,x_3\}}(x)$
0 0 0 0 0	0	0	0	0	0	0
0 0 0 0 1	0	0	0	0	1	1
0 0 0 1 0	0	0	0	1	0	1
0 0 0 1 1	0	0	0	1	1	1
0 0 1 0 0	0	0	1	0	0	1
0 0 1 0 1	0	0	1	0	1	1
0 0 1 1 0	0	0	1	1	0	1
0 0 1 1 1	0	0	1	1	1	1
0 1 0 0 0	0	1	0	0	0	0
0 1 0 0 1	0	0	0	1	1	1
0 1 0 1 0	0	0	1	1	0	1
0 1 0 1 1	0	0	1	1	1	1
0 1 1 0 0	0	0	1	0	1	1
0 1 1 0 1	0	0	1	1	1	1
0 1 1 1 0	0	0	1	1	1	1
0 1 1 1 1	0	0	1	1	1	1
1 0 0 0 0	1	0	0	0	0	0
1 0 0 0 1	1	0	0	0	1	1
1 0 0 1 0	1	0	0	1	0	1
1 0 0 1 1	1	0	0	1	1	1
1 0 1 0 0	1	0	1	0	0	1
1 0 1 0 1	1	0	1	0	1	1
1 0 1 1 0	1	0	1	1	0	1
1 0 1 1 1	1	0	1	1	1	1
1 1 0 0 0	1	1	0	0	0	0
1 1 0 0 1	1	0	0	1	1	1
1 1 0 1 0	1	0	1	1	0	1
1 1 0 1 1	1	0	1	1	1	1
1 1 1 0 0	1	0	1	0	1	1
1 1 1 0 1	1	0	1	1	1	1
1 1 1 1 0	1	0	1	1	1	1
1 1 1 1 1	1	0	1	1	1	1

Proposition 4 *Let us consider a system S whose m files are infectible by a n-form polymorphic virus. Let us assume that $n < m$. Then the transition graph of the infection by the virus $(x_{v_1}, \ldots, x_{v_n})$ contains a directed complete subgraph whose points are the points $(e_{v_i})_{1 \le i \le n}$.*

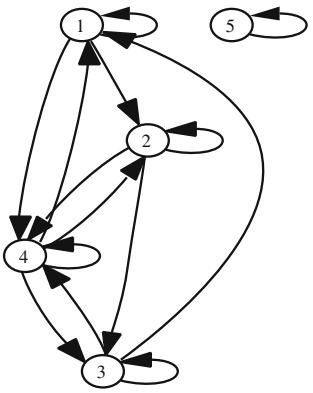


Fig. 6 Partial transition graph of a 3-form polymorphic virus

Proof Without loss of generality, we suppose that the mutation process is done according to the following scheme $x_{v_i} \to x_{v_{i+1}}$ with $x_{v_n} \to x_{v_1}$. Let us note x_{f_i}, for $1 \le i \le m$ the variable corresponding to files that can be infected by the virus.

To prove the proposition, it suffices to show that every variable x_{v_i} is contained in the (disjunctive or algebraic) normal form of every coordinate function F_j^t, for $v_1 \le j \le v_n$. We will consider the algebraic normal form (ANF) [4]. Let us recall that the ANF of a N-variable Boolean function f is described by

$$f(x_1, x_2, \ldots, x_N) = \sum_{\alpha \in \mathbb{F}_{2^N}} a_\alpha x^\alpha \quad a_\alpha \in \mathbb{F}_2$$

where $x = (x_1, x_2, \ldots, x_N)$ and $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_N^{\alpha_N}$ with $\alpha_i \in \mathbb{F}_2$. The ANF coefficients a_α are computed by means of the Möbius transform [4]:

$$a_\alpha = g(\alpha) = \bigoplus_{\beta \preceq \alpha} f(\beta_1, \beta_2, \ldots, \beta_N),$$

where $\beta \preceq \alpha$ is the partial ordering on the subset lattice of $\mathbb{F}_2^N$. In other words, $\beta \preceq \alpha \Leftrightarrow \beta_i \le \alpha_i \forall i$.

With these notations, let us show that the ANF of every coordinate function F_j^t, with $v_1 \le j \le v_n$ contains all the variables x_{v_i}, $1 \le i \le n$. Let us denote $M = m + n$.

Let us consider the M-tuple $\alpha_i = (\ldots, x_{f v_{j-1}}, \ldots, x_{f v_1}, 0, \ldots, x_{v_i}, 0, \ldots, 0)$ made of $j - 1$ non null elements (which correspond to files that can be infected by the virus) and of a single non null element which corresponds to the

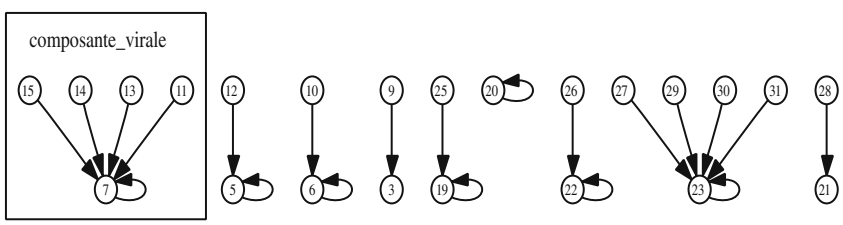


Fig. 5 Partial iteration graph of a 3-form polymorphic virus

mutated form v_i of the virus. Let us show that the coefficient $a_{\alpha_i} = 1$ in the ANF of F_j^t for $v_1 \leq j \leq v_n$.

By construction of the function F^t, the M-tuple $\beta = \alpha_i$ is the single one such that $F_j^t(\beta) = 1$. Indeed, in order to generate the mutated form v_j, at least $j-1$ infectible files must be present.

Since the number of M-tuples β for which $F_j^t(\beta) = 1$, is odd, for every j ranging from v_1 to v_n, we have $a_{\alpha_i} = 1$ for every j ranging from v_1 to v_n. Consequently, whatever maybe the variable x_{v_i} (by iterating on the index i), it is represented in the ANF of every coordinate function which corresponds to a mutated form of the virus. Hence the result. □

Remark For most practical polymorphic viruses, the number of forms n is far larger than the number of file to infect in S. Thus the complete subgraphs does only partially exist. Nonetheless, this general result states in a different manner the result of D. Spinellis about polymorphic virus detection complexity [13].

Theorem 1 *Let S be a system containing m infectible files with respect to a n-form polymorphic virus and let us assume that $n < m$. Detection of a polymorphic virus, when based on the interactions between its mutated forms is NP-complete.*

Proof The proof uses the fact that determining whether there exists a clique of size n (complete sub-graph) in a graph (the transition graph) is a NP-complete problem [10]. In the case of our theoretical model (see Convention 1), solving this problem may prove to be easy—there is only one connected component, which is not reduced to a single point and which contains a complete sub-graph of size n with respect to the virus $(x_{v_1}, x_{v_2}, \ldots, x_{v_n})$. In a real-life context (existence of many interactions between files that can be infected and those that cannot), there may exist many other connected components from one side, and the viral component is not necessarily limited to the viral files only. In other words, in a real context, the transition graph is so "complex" that it represents a hard instance of the clique problem. Hence the result. □

Other results have been established for polymorphic/metamorphic viruses from our Boolean model [6], especially on the structure of the iteration graph.

It can be stated for polymorphic/metamorphic viruses that compared to simple virus (singleton viral set), the "viral information" is no longer concentrated on a unique file but on the different possible mutated form, even if every form is able to spread the infection alone. As far as the equivalence relation $\mathcal{R}_v$ is concerned, there is no redundancy between the mutated forms. But if any of them is being analysed, the mutation process will be known and every possible evolved form will be detected (at least theoretically). We thus are facing the same situation as in a singleton viral set. The weakness comes from the fact that the virus'action is totally defined in the code, whatever may be its form.

4 *K*-ary codes

Our model enables to prove that simple viruses (singleton viral sets) and polymorphic/metamorphic viruses (largest viral sets) are somehow equivalent. Their respective iteration graphs are quite similar, up to the choice of the equivalence relation defined in the Sect. 2.1 [6]: as an example there does not exist paths of length greater than 2 in both cases. That precisely means that all the viral information is available after a single infection step.

The essential interest of k-ary viruses precisely comes from the fact that the viral information is scattered over different files. Consequently, we can define new classes of viruses that are not equivalent to existing viruses. Two main classes of k-ary codes have been identified:

- *Class I codes*. These codes are working sequentially (one after the another). Three subclasses are to be considered:
  - *A subclass (dependent sequential codes)*. Every part refers or contains a reference to the other ones. It is the weakest class in term of detection since successful detection of one part helps to detect the others;
  - *B subclass (independent sequential codes)*. No part is referring to another one. Detecting one part does not endanger the other ones. The detected part may be automatically replaced under a different form.
  - *C subclass C (weakly dependent sequential codes)*. Dependency between codes is partial and directed only.
- *Class II codes*. These codes are working in parallel. The same three previous subclasses are to be considered in the same way. By definition, the k parts have to be present and active at the same time in the system. This may represent a weakness unless efficiently designed and implemented.

4.1 *K*-ary codes in sequential mode

By definition, these codes are in fact a partition of k parts. Each of them performs only a chunk of the whole action, according to a sequential mode. Each of them must look like an innocuous file. The offensive (malicious) action can be determined by simultaneously considering the k parts only, or at least a subset of a fixed size. Many different settings can be imagined.

Only a few very trivial cases are known, with $k = 2$ and all belong to the A or C subclasses. Due to their inherent weaknesses, they are all bound to be detected. We can quote the

Perrun virus or the *Scob/Padodor* Trojan. We could also mention the case of the OpenOffice virus called *Final_Touch* [3].

We now are going to detail the case of a B subclass k-ary worm. Up to the author's knowledge, this type of code has never been found yet "in the wild". In order to thoroughly study this technology, we have designed and analysed a proof-of-concept worm, called POC_SERIAL. We will first present the worm itself and then we will consider the general model for this subclass.

The POC_SERIAL **worm** We present here a generic setting of the worm only. Our aim is to focus on the algorithmic aspects but many variants are possible. Moreover for both ethical and legal reasons, we will not give implementation details. Besides the fact that it is not useful for the understanding, this kind of technology cannot be detected by existing antiviral technologies.

The POC_SERIAL worm is made of k parts. From a practical point of view, we can choose $4 \leq k \leq 8$. Every of these parts are totally indistinguishable from any other legitimate, innocuous program, especially as far as its behaviour is concerned. The final payload consists in stealing a document and making it escape from the target machine.[2]

Let us denote $(P_i)_{1\leq i\leq k}$ the constituent parts of the POC_SERIAL worm (the worm will be noted W_k for short). Without loss of generality, we assume that these parts are working according to the following order: $P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_k \rightarrow P_1$. The main features of W_k are:

- at time instant t, only part P_i is present in the system. To manage the worm evolution in the system, the following rules have been adopted (among many other possible), assuming that $k + 1 = 1$:
  1. if P_i and P_{i+1} are simultaneously present in the system, then P_i wipes out itself;
  2. if P_i is present but P_{i+1} did not then P_{i+1} is created and P_i wipes out itself;

  These two rules are iteratively applied until obtaining a k-tuple of weight 1. It is essential to note that the first rule should never used when considering an efficient implementation of W_k. The critical point lies in the fact that only one part must be present in the system at a given time instant;
- for every $1 \leq i \leq k$, the file P_i does never refer to file P_j ($1 \leq i \leq k$ et $i \neq j$). In other words, the analysis of any P_i does not enable to guess what the other P_j are. Nonetheless, every part must be able to detect the existence in the system of any other part without any relevant knowledge on it and without containing any reference to it. This point is critical too. Techniques like those presented in [11] are very useful at achieving this constraint (among many other possible ones);
- from a practical point of view, a minimal time delay $\Delta t > 0$ is introduced between the respective action of P_i and P_{i+1} in order to make sure that any antivirus cannot have the two files simultaneously at its disposal;
- every part P_i brings deception techniques into play. The latter aim at creating fake interactions within the target system. These techniques are very useful as Theorem 2 will prove it.

According to the worm variant, the infection spreads either from a fixed server (centralised mode) or from any target machine directly (decentralised mode).

The theoretical model For sake of simplicity, we will consider the case of $W_4 = (P_1, P_2, P_3, P_4)$. Moreover, we will limit the use of the Boolean variables to the four parts of the worm. From a general point of view, we would extend $\mathbb{F}_2^4$ to $\mathbb{F}_2^n$ to take every file in the system into account. Let us first consider the context of the simplified model (refer to Convention 1). Under the working rules defined in the previous section, transition and iteration functions for W_4 are given in Table 4. Each part P_i is denoted by variable x_i. The iteration graph is given in Fig. 7 and the transition graph in Fig. 8. Let us state a first combinatorial result on the iteration graph.

Proposition 5 *Let S be a system containing N files before infection. The iteration graph modelling the infection with respect to W_k (simplified model) has:*

Table 4 Transition and infection function of the W_4 worm

x_4	x_3	x_2	x_1	$F_4^t(x)$	$F_3^t(x)$	$F_2^t(x)$	$F_1^t(x)$	$f_{\{x_1,x_2,x_3,x_4\}}(x)$
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	1	0	1	0	0	0
0	1	0	0	1	0	0	0	0
0	1	0	1	1	0	0	0	0
0	1	1	0	1	0	0	0	0
0	1	1	1	1	0	0	0	0
1	0	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0	0
1	0	1	0	0	1	0	0	0
1	0	1	1	0	1	0	0	0
1	1	0	0	0	0	0	1	0
1	1	0	1	0	0	1	0	0
1	1	1	0	0	0	0	1	0
1	1	1	1	0	0	0	1	1

[2] We use a "real" payload in order to evaluate resistance to behaviour-based detection.

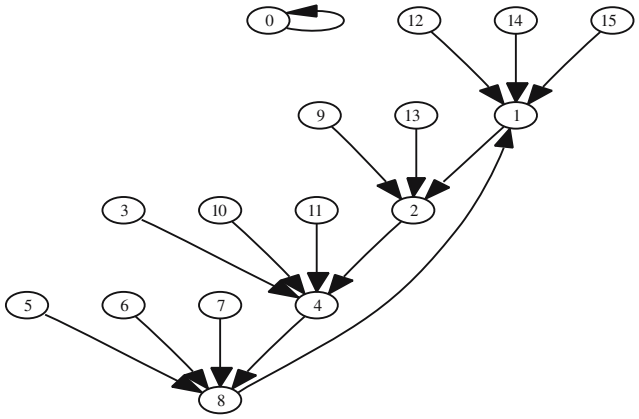


Fig. 7 Iteration graph of the W_4 worm

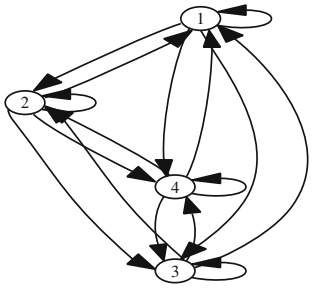


Fig. 8 Transition graph for the W_4 worm

- 2^N *connected components, each being reduced to a single fixed point;*
- 2^N *connected components, each of them containing* $2^k - 1$ *points and a cycle of length exactly* k.

Proof Let us denote $W_k = (x_k, x_{k-1}, \ldots, x_1)$. Without loss of generality, let us describe the files in the system S that are different from any viral part W_k, by the N-tuple $(x_{N+k}, x_{N+k-1}, \ldots, x_{k+1})$. Every point $x \in \mathbb{F}_2^{N+k}$ will be described by the couple $(y, \overline{y})$ with $\overline{y} \in \mathbb{F}_2^k$ ("viral" part of x).

For the single point components, they each contain a point $x \in \mathbb{F}_2^{N+k}$ whose support does not contain any index relatively to a file which would be any part P_i of the code W_k. There are exactly 2^N such points.

For the second point of the proposition, every connected component contains every points whose image given by the function F^t is one of the k points e_i (recall that this point only the i-th coordinate which is non-zero). Moreover, for every point e_i, we have $F^t(e_i) = e_{i+1}$ with the convention that $k + 1 = 1$. The file that cannot be infected by the virus have no interaction with any of the part of the code W_k, for all $x = (y, \overline{y})$. Then we have $F^t(x) = (y, F^t(y'))$.

Besides that, for $x = (y, \overline{y})$ and $x' = (y', \overline{x'})$ such that $y \neq y'$, we have $F^t(x) \neq x'$ and $F^t(x') \neq x$. Since there are 2^N possible y, we have the result. □

From that proposition, we now can give the detection complexity for k-ary sequential code, when considering functional analysis (in other word file interactions).

Theorem 2 *Let S be a system which has been infected by a k-ary sequential code. Detecting this code, when relying on functional evolution analysis is NP-complete.*

Proof Let us now consider the generalised model. In other words we take every possible file interaction in S into account. This implies that the connected components are no longer disconnected one from the another. There exist many edges that bridge connected components. Those bridges may be build by deception techniques to increase the graph complexity (see subsection devoted to the POC_SERIAL worm).

Detection of W_k-like codes, by definition, implies to detect the whole set of its parts. For every $x \in \mathbb{F}_2^{N+k}$, we have to search for a circuit of length exactly k which walk through an arc whose endpoints are any two viral parts. It has been proved by Thomassen [14] that it is a NP-complete problem, in the general case. Hence the result. □

It is worth mentioning that this results could have been proved in a similar way by considering, among many others, either the problem of partial sub graphs isomorphism [8,10] or the problem of determining whether a graph has stability number at most s (in our case $s = 2^{k-1}$) [1, pp.40].

Thus detecting sequential k-ary codes is untractable from a general point of view. The practice can be made very close from the theory by considering suitable values of k and Δt. When considering the transition graph, we get the same complexity result. Contrary to polymorphic/metamorphic codes, the analysis of any of the parts P_i is not sufficient in itself: we must have all the parts at one's disposal. Any part contains a subset of the viral instructions only. This can be stated as follows.

Proposition 6 *Let $F^t(x)$ be the transition function modelling the infection by a sequential k-ary code of B subclass (W_k-like) in a N-file system S (before infection). Let f_{W_k} be the corresponding infection function. Then, for every $u \in \mathbb{F}_2^k$, such that $supp(u) \subseteq supp(W_k)$, we have*

$$P[(f_v \circ F^t)(x) =< x, u >] = \frac{1}{2} - s(u)\frac{1}{2^k},$$

with $s(u) = (-1)^{wt(supp(u))}$.

Proof Let us recall that $\mathrm{supp}(W_k)$ describes the $(N+k)$-tuple in which only k coordinates which corresponds to the constituent parts of W_k equal 1. The result can be easily proved by considering the fact that $< x, u >= 0$ with a probability exactly equal to $\frac{1}{2}$ and that the function $(f_v \circ F^t)(x) = 1$ for $x \cap \mathrm{supp}(W_k) = \mathrm{supp}(W_k)$. Then for that last equality, either $\mathrm{supp}(u)$ has an odd weight and thus we have $< x, u >= 1$ and $s(u) = -1$, or on the contrary $< x, u >= 0$ and $s(u) = 1$. Hence the results. □

This essential result shows that sampling-based statistical detection on the transition graph becomes itself untractable as soon as k increases.

4.2 K-ary codes in parallel mode

These codes are made of k memory resident parts which act in parallel. This seems to be a weakness since the analysts has at his disposal the whole viral information, at least theoretically. Nonetheless, from a practical point of view, k-ary codes in parallel mode can prove to be very powerful. Moreover, they represent a critical risk that antivirus are likely to fail to fight against. Two main reasons for that claim, may be considered:

- these codes may be deployed by stealth techniques or even worse by sophisticated, virtualisation-based rootkit techniques like *BluePill* [12] and *SubVirt* [9];
- detecting is not sufficient in itself. Disinfection is at least as important as detection. We will see how a k-ary parallel code of A or C subclasses can hold antivirus software in check with respect to disinfection.

4.3 The PARALLEL_4 and PARALLEL_4_S virus

This code has been found in the wild in 2004 but up to the author's knowledge it is not documented yet. Only a few antivirus software are able to detect it as a generic code. But the essential feature of this code is that antivirus are totally unable to eradicate it from infected computers. Detection after detection, the code remains active. This code has been named PARALLEL_4 according to its working mechanisms. Indeed, it is a A subclass 4-code in parallel mode. We are now going to present a sophisticated proof-of-concept variant that we derived from the original code. This variant is called PARALLEL_4_S

The main features of PARALLEL_4_S are the following:

- every of the four parts contains a chunk of the viral information only;
- the four parts are simultaneously active in memory;
- every part is able to regenerate the three others, under an evolved form (the mutation process includes the name of the code/process itself);
- besides its inherent assigned action, every part keeps watch on the three others and check whether it is still active or not. If one or more are no longer resident (the process has been killed), a random one of three others regenerates it. Since the names of the four processes are likely to frequently change, every part is able to blindly identify the others;
- every part, at irregular time instant, self-refreshes while mutating its name in order to simulate a normal memory activity, at the process level.

Every time an antivirus succeeds in detecting one the four parts, the three others are regenerating it under different name and form. Consequently, disinfection is no longer possible.

4.4 The theoretical model

From formal point of view, k-ary parallel codes are not different from sequential k-ary codes, whatever may be the subclass. Processors work sequentially by nature (Turing machines) so both codes are somehow equivalent.

The main result that can be stated here follows.

Proposition 7 *Le S a system which is infected by a parallel k-ary code, called P_k. Then the corresponding transition graph which describes the infection of S with respect to $P_k = (x_k, x_{k-1}, \ldots, x_1)$ (simplified model) contains a complete subgraph whose nodes are points $(e_i)_{1 \leq i \leq k}$.*

Proof The proof is essentially the same as for k-ary sequential codes. □

When considering deception techniques and the Boolean generalised model, Proposition 7 enables to prove that detecting parallel k-ary codes is untractable in the general case.

Theorem 3 *Let S a system which is infected by a parallel k-ary code. File interaction-based detection of this code in S is a NP-complete problem.*

The proof is the same as that given for Theorem 2. From a general point of view, most of the theoretical results we got for sequential k-ary codes hold for parallel k-ary codes as well.

5 Future work and conclusion

K-ary codes are a powerful family of codes. Besides a huge number of beneficial applications (e.g. protection against software piracy), they represent a huge risk in term of malware. Existing antivirus technologies are totally inefficient at detecting those codes as our study and experiments have confirmed it. It would be wrong to imagine that any technical protection against these codes is tractable due to the average complexity of their detection. Detection has to face combinatorial problems that cannot be solved in an amount of time that is compatible with commercial antivirus software.

Theoretical analysis of k-ary codes is at its early stage only. Much has still to de done. Vector Boolean functions (transition functions) enable to formalise things in a powerful way. But, in this paper we consider only a few possible functions. New building rules can be considered that will give birth to new kind or behaviour of k-ary codes. As an example, we could define rules in order to get Hamilton graphs or graphs containing cliques both for the transition graph and the iteration graph.

However, due to their deterministic nature, vector Boolean functions can grasp only a small subset of k-codes. Indeed, for $x \in \mathbb{F}_2\ n$, the value $F^t(x)$ is then fixed: the resultant k-ary code is deterministic too. Consequently, we could extend our model by considering non-deterministic automata (thus any $x \in \mathbb{F}_2\ n$ may have more than one successor). First studies [6,7] have shown that this enables to design malware whose detection is far beyond the NP-complete class, not to say undecidable. Moreover, non-deterministic automata make possible to consider transition functions are not defined on a finite number of x.

References

1. Bondy, J.A.: Basic Graph Theory: Paths and Circuits. In: Graham, R., Grötschel, M., Lovasz, L. (eds) Handbook of Combinatorics, Vol. 1, North Holland (1995)
2. Cohen, F.: Computer viruses, PhD Thesis, University of Southern California (1986)
3. De Drézigué, D., Fizaine, J.-P., Hansma, N.: In-depth analysis of the viral threats with OpenOffice.org documents. J. Comput. Virol. **2**(3), 187–210 (2006)
4. Filiol, E.: Techniques de reconstruction en cryptographie et en théorie des codes. PhD Thesis, École Polytechnique (2001)
5. Filiol, E.: Computer viruses: From theory to applications. IRIS International Series. Springer, France, ISBN 2-287-23939-1 (2005)
6. Filiol, E.: Techniques virales avancées. IRIS Series. An English translation is pending (due mid-2007). Springer, France (2007)
7. Filiol, E.: Metamorhism, formal grammars and undecidable code mutation. Int. J. Appl. Math. Comput. Sci. **4**(2), 503–508 (2007). http://www.waset.org/ijamcs/v4-2.html
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman (1979)
9. King, S.T., Chen, P. M., Wang, Y.-M., Verbowski, C., Wang, H.J., Lorch, J.R.: SubVirt: Implementing malware With virtual machines, University of Michigan et Microsoft Research (2006)
10. Papadimitriou, C.H.: Computational complexity. Addison-Wesley, Reading. ISBN 0-201-53082-1 (1995)
11. Riordan, J., Schneier, B.: Environmental key generation towards clueless agents. In: Vigna, G. (ed.) Mobile Agents and Security Conference'98. Lecture Notes in Computer Science, pp. 15–24, Springer, Heidelberg (1998)
12. Rutkowska, J.: Subverting vista kernel for fun and profit. SysCan'06 Conference, Singapore, July 21 2006
13. Spinellis, D.: Reliable identification of bounded-length viruses is NP-complete. IEEE Trans. Inf. Theory **49**(1), 280–284 (2003)
14. Thomassen, C.: Even Cycles in Directed Graphs, European Journal in Combinatorics, **6**, 85–89 (1985)
15. Zuo, Z., Zhou, M.: Some further theoretical results about computer viruses. Comput. J. **47**(6), 627–633 (2004)