Entropy **2014**, *16*, 6602-6623; doi:10.3390/e16126602

OPEN ACCESS
entropy
ISSN 1099-4300
www.mdpi.com/journal/entropy

Article

Detection and Modeling of Cyber Attacks with Petri Nets

Bartosz Jasiul [1,*], Marcin Szpyrka [2] and Joanna Śliwa [1]

[1] C4I Systems' Department, Military Communication Institute, ul. Warszawska 22A, 05-130 Zegrze, Poland; E-Mail: j.sliwa@wil.waw.pl

[2] Department of Applied Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland; E-Mail: mszpyrka@agh.edu.pl

* Author to whom correspondence should be addressed; E-Mail: b.jasiul@wil.waw.pl; Tel. +48-22-6885592.

External Editors: James J. Park and Wanlei Zhou

Received: 30 October 2014; in revised form: 4 December 2014 / Accepted: 16 December 2014 / Published: 19 December 2014

Abstract: The aim of this article is to present an approach to develop and verify a method of formal modeling of cyber threats directed at computer systems. Moreover, the goal is to prove that the method enables one to create models resembling the behavior of malware that support the detection process of selected cyber attacks and facilitate the application of countermeasures. The most common cyber threats targeting end users and terminals are caused by malicious software, called malware. The malware detection process can be performed either by matching their digital signatures or analyzing their behavioral models. As the obfuscation techniques make the malware almost undetectable, the classic signature-based anti-virus tools must be supported with behavioral analysis. The proposed approach to modeling of malware behavior is based on colored Petri nets. This article is addressed to cyber defense researchers, security architects and developers solving up-to-date problems regarding the detection and prevention of advanced persistent threats.

Keywords: malware; cyber attack; colored Petri net; malware detection; behavioral analysis

---

1. Introduction

According to the numerous cyber security reports, the number of cyber threats is increasing rapidly from 23,680,646 in 2008 [1] to 5,188,740,554 in 2013 [2]. This is nowadays one of the most vexing

problems in computer systems security [3]. At the end of 2013, Kaspersky Lab, the Russian producer of anti-virus software, reported that it currently detects and blocks more than 315,000 new malicious programs every day, a significant increase from 2012, when 200,000 malicious programs were detected and blocked each day on average.

According to Nomura Research Institute annual report on cyber security trends, in 2012, 100% of organizations had anti-virus products installed [4]. Despite this, according to this report, about 30% of organizations are systematically infected by malware. The reason for this situation is not, as it might be expected, the inappropriate update of operating systems and virus definition files, but the lack of all signatures for existing and appearing threats.

A major group of malware is composed of existing (known) parts of malicious codes [5]. Regardless of the fact that their installation packs are different, this make them similar in a way that they may launch applications with the same features. This simplicity of malicious code development and the effectiveness of obfuscation mechanisms [6,7] available for the attackers nowadays make them armed with a powerful weapon.

Moreover, according to the study conducted in 2012 by the Verizon Research, Investigations, Solutions, Knowledge (RISK) Team in cooperation with many national federal organizations, including the Australian Federal Police, Irish Reporting and Information Security Service, and United States Secret Service, the following findings were made [8]:

- 54% of malware took months to discover,
- 29% of malware took weeks to discover,
- 13% of malware took days to discover.

This report shows how important it is to introduce new techniques that speed up the process of malware detection to hours. The authors of the report [4] indicate that anti-virus products should be supported by malware behavioral analysis tools in order to detect those attacks for which signatures were not established yet. An existing example of an application that uses behavioral analysis for advanced persistent threat detection is Digital DNA by HBGary, which extends the capabilities of McAfee Total Protection anti-virus [9]. The detailed technical specification of this solution has not been released for the public yet. The product brochure reveals the information that multiple low level behaviors are identified for every running program or binary. This leads to the conclusion that each application is observed from a behavioral perspective. McAfee is proud that the solution allowed the detection during the last year of more zero-day attacks than the previous five years combined. This indicates the scale of new malware development and the efficacy of the behavioral approach.

2. Motivation

An overwhelming number of computer systems are connected to each other by a global network, the Internet, which allows producing results beyond those achievable by the individual systems alone. The outcomes of cooperative work and the accessibility of information are perceived and appreciated probably by all of its users. The advantages of this technology are available, unfortunately, also for hostile goals, which was highlighted in the Introduction section.

Although awareness of necessary security applications seems to be common and the tools used for that purpose are getting more and more advanced, the number of successful attacks targeted at computer systems is growing [10]. They are mostly related to denial of offered services, gaining access to or stealing private data, financial fraud, *etc.* [11,12]. Moreover, the evolution towards cloud computing, increasing use of social networks, mobile and peer-to-peer networking technologies, which are an intrinsic part of our daily routine, bring many conveniences to our personal life, business and government, gives the possibility of using them as tools for cyber criminals [13] and a potential path of malware propagation [14,15].

Cyber criminals are focused on finding a way to bypass security controls and gaining access into the protected network for many reasons. These may be gathering sensitive data or money fraud. That is why organizations, companies, governments and institutions, as well as ordinary citizens all over the world are interested in the detection of all attempts of malicious actions targeting their computer networks and single machines.

The success rate of the applied methods for malware detection depends on the reliability of the malware models. Usually, they are based on the code signatures. Security controls (e.g., anti-virus tools) might be unadjusted, because the signatures of new threats have not been identified yet. Hackers often use existing parts of code in order to implement new types of malware. This allows one, in turn, to quickly develop the signatures of new dangerous software. Therefore, the more signatures that are deployed, the more malicious codes that are identified. On the other hand, one of the methods of misleading the signature-based detection systems is code obfuscation [16], the aim of which is generating, from already existing code, a new application that cannot be assessed yet as being risky by security controls [17]. This technique is simple in application and potentially successful, so that also successful countermeasures are necessary. One of the examples is to follow the behavior [18] of the malicious software in order to identify it and eliminate it from the protected system.

In order to overcome the limitations of signature-based anti-virus tools, there are various modeling methods of malware behavior proposed by the scientific community. As opposed to signature-based techniques, which analyze the static contents of the binaries, behavior techniques are focused on run-time events caused by those binaries observed on different levels (processor, operating system), which form sequences of related actions—so-called patterns. It is very common that sensors focused on monitoring system calls are used to identify the symptoms of malware behavior. From this vast amount of apparently independent events, it is necessary to identify symptoms that match a unique pattern. This approach relies heavily on the model of malware behavior that was used as the basis of a particular detection method.

In [19,20], we can find the analysis of both groups and sequences of system calls in the form of n-gram models. These approaches investigate the diversity of system calls by verification of a large-scale collection of system events that were related to the activities performed by particular hosts' users. N-gram models enable one to identify similarities among different types of malware; however, the matching process is not strictly defined and may be implemented differently.

Another approach was proposed in [21], which describes an individual system call analysis that tracks processor operations on the assembler code level on the basis of control flow graphs. The case study presented was based on malware affecting web browsers and any loaded browser helper object. With the use of this method, the authors were able to handle sensitive user information, such as the URLs that

a user visits or the content of the web pages that are loaded in Microsoft Internet Explorer. This way, a leakage of sensitive information outside the browser can be stopped.

Similarly to [21], a system call dependency graph [22] is a technique for extracting optimally discriminative specifications, based on graph mining and concept analysis, which uniquely identifies a class of programs. Such a discriminative specification technique scales to large classes of programs due to probabilistic sampling of the set of events. This technique for mining behaviors by contrasting two sets for malicious and benign programs relies on a precise representation of software behavior, which could be unsuccessful for zero-day attacks constructed from existing parts of malware, which might behave differently.

In [23], we can find an interesting proposal of the bounded feature space behavior modeling (BOFM) framework for scalable malware detection. BOFM models the interactions among software (which can be malware or benign) and security-critical OS resources in a scalable manner. Information collected at run-time (e.g., based on API hooking) according to this model is then used by machine learning algorithms to learn how to accurately classify software. This approach enables one to perform system monitoring during the user's regular interaction with the system. However firstly, the learning phase needs to be performed. This approach may suffer from additional false positive and negative detection rates depending on the learning set, as well as the learning process.

In [24] modeling of the propagation of email worms was proposed. The proposed model can precisely present the repetitious spreading process caused by malicious software distributed as email attachments. This method can be used to evaluate the infection process of newly-created malware, so called zero-day attacks. It could be useful to alarm the users of some network environments (e.g., company, county, region); however, it would not be sufficient for malware detection.

Some of the methods presented above describe only the model used to describe malware behavior [20,23,24]; however, others focus more on the detection method [21,23]. They can detect malware activities on the processor level [21] or the operating system calls [19] in run-time [23] or in a controlled or emulated environment [24]. The main assumption taken for our work was that the malware tracking tool needs to operate during the normal operation of the host machine on the basis of events observed by the operating system monitor sensor. The model must also be able to resemble the behavior of different kinds of malware (not focusing only on Internet browser spyware or e-mail worms) and efficiently support the process of detection. It was also important to provide the possibility to use a particular classifier that can improve the success rate of the detection process (minimizing the false positive and false negative rates).

The aim of this article is to present an approach to the modeling and detection of malware behavior with the use of formal models (classifiers) based on colored Petri nets (CP-nets [25,26]). We prove that the method enables one to create models resembling the behavior of malware, and these models support the detection of cyber threats directed at computer systems.

3. CP-nets

Colored Petri nets form a discrete-event modeling language combining the capabilities of Petri nets [27] with the capabilities of a high-level programming language. CP-nets provide graphical notation typical for Petri nets, but net elements are described using the CPN ML programming language [25].

A non-hierarchical CP-net [25] is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where:

(1) P is a finite set of places.

(2) T is a finite set of transitions, such that $P \cap T = \emptyset$.

(3) $A \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs.

(4) Σ is a finite set of non-empty color sets.

(5) V is a finite set of typed variables, such that $Type[v] \in \Sigma$ for all variables $v \in V$.

(6) $C\colon P \to \Sigma$ is a color set function.

(7) $G\colon T \to Expr_V$ is a guard function, such that $\forall_{t \in T}\, Type[G(t)] = Bool$, where $Expr_V$ denotes the set of expressions (possibly with free variables from set V) provided by CPN ML.

(8) $E\colon A \to Expr_V$ is an arc expression function, such that $\forall_{a \in A}\, Type[E(a)] = C(p)_{MS}$, where p is the place connected to the arc a and X_{MS} denotes the set of multisets over set X.

(9) $I\colon P \to Expr_\emptyset$ is an initialization function, such that $\forall_{p \in P}\, Type[I(p)] = C(p)_{MS}$.

A marking (state) of a CP-net is a function M defined on the set of places P, such that $\forall_{p \in P} M(p) \in C(p)_{MS}$. The initial marking is obtained by evaluating the initialization expressions.

An execution of a CP-net is described by means of an occurrence sequence. It specifies the markings that are reached and the transitions that occurred. To make it possible to evaluate arc expressions, it is necessary to assign (bind) some values to free variables occurring in arc expressions on the arcs connected to the transition and in the transition guard. A transition is enabled (ready to occur) if it is possible to construct such a binding that the guard evaluates as true and each of the arc expressions evaluate as tokens, which are present on the corresponding input places. An occurrence of a transition t removes tokens from input places of t and adds tokens to its output places. The multisets of removed/added tokens are specified by the expressions of the corresponding arcs.

Hierarchical CP-nets are composed of non-hierarchical modules. Substitution transitions and fusion places are used to combine such modules. The former idea allows the user to refine a transition and its surrounding arcs into a more complex net, which usually gives a more precise and detailed description of the activity represented by the substitution transition. A fusion of places allows users to specify a set of places that should be considered as a single one. This means that they all represent a single conceptual place, but are drawn as separate individual places (e.g., for clarity reasons). For more details, see [25].

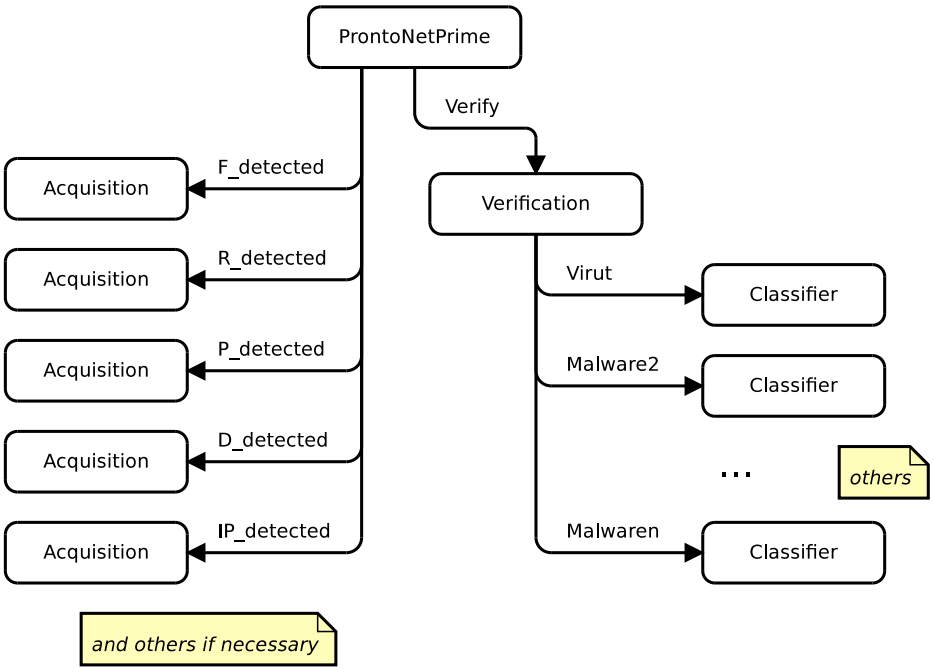
4. CP-net Malware Models

Malware executes functions and operations on the same system resources as legitimate applications. Therefore, it is possible to identify patterns that can be used as models of software behavior. These are potentially: (i) operations on files; (ii) operations on the system registry; (iii) run processes and applications; (iv) communication with specific IP addresses; and (v) communication with domains. These actions can be malicious and performed among many actually harmless ones.

The malware detection takes as an input a set of suspicious events received from the process' hooking engine, so-called PRONTOlogy, developed by the authors of this article [28,29]. This engine is based on ontology reasoning [30] used for the purpose of filtering single malicious incidents among hundred of thousands of regular ones. These single events are passed on to the PRONTOnet engine, the operation of which is based on formal CP-net malware models, for further investigation.

The PRONTOnet uses formal malware models and performs the detection process by passing (moving) through particular places in the model, using the CP-net vocabulary and characteristics described above. CP-nets are one of the most widespread classes of Petri nets. They provide an easy to understand graphical notation similar to other commonly-used graphical languages, e.g., some UML diagrams. The description language is also easy to learn. It is reduced to a small number of statements typical for high-level programming languages. The semantics of CP-nets is formally defined, and a reach set of analysis methods for CP-nets is accessible. In comparison with other formals methods, CP-nets equally describe the states and actions of the modeled system. Depending on the needs, we can focus on the former or the latter of these aspects. Moreover, the mechanisms of the hierarchical models' design enable users to construct complex models and/or to swap some parts of a model if necessary.

Figure 1. Colored Petri net (CP-net) model of PRONTOnet : Page hierarchy graph.



The hierarchical structure of the CP-net model defined by the authors is shown in Figure 1. The detection process progresses on the basis of the levels of the hierarchy. The primary module of the CP-net

model, representing the PRONTOnet threat tracking tool, is depicted in Figure 2. On the left-hand side of this figure, there is a column of places, marked with ellipses, storing tokens that represent particular assets that might be affected by malware: F, a place storing tokens indicating files; R, registry entries; P, processes; D, domains; IP, IP addresses that the malware may communicate with. Markings of these places represent tracked symptoms. The second column in Figure 2 is composed of substitute transitions that are related to the acquisition process depicted in Figure 3. The next column is made up of places indicating particular assets affected by malware activated in the monitored system. Markings of these places represent observed symptoms. They are processed by a substitute transition called Verify, which is aimed at reasoning if the system is infected by a certain malware type. As a consequence, the place RESULT is marked with a vector informing about particular malware type and related symptoms.

Figure 2. CP-net model of PRONTOnet: Primary module.

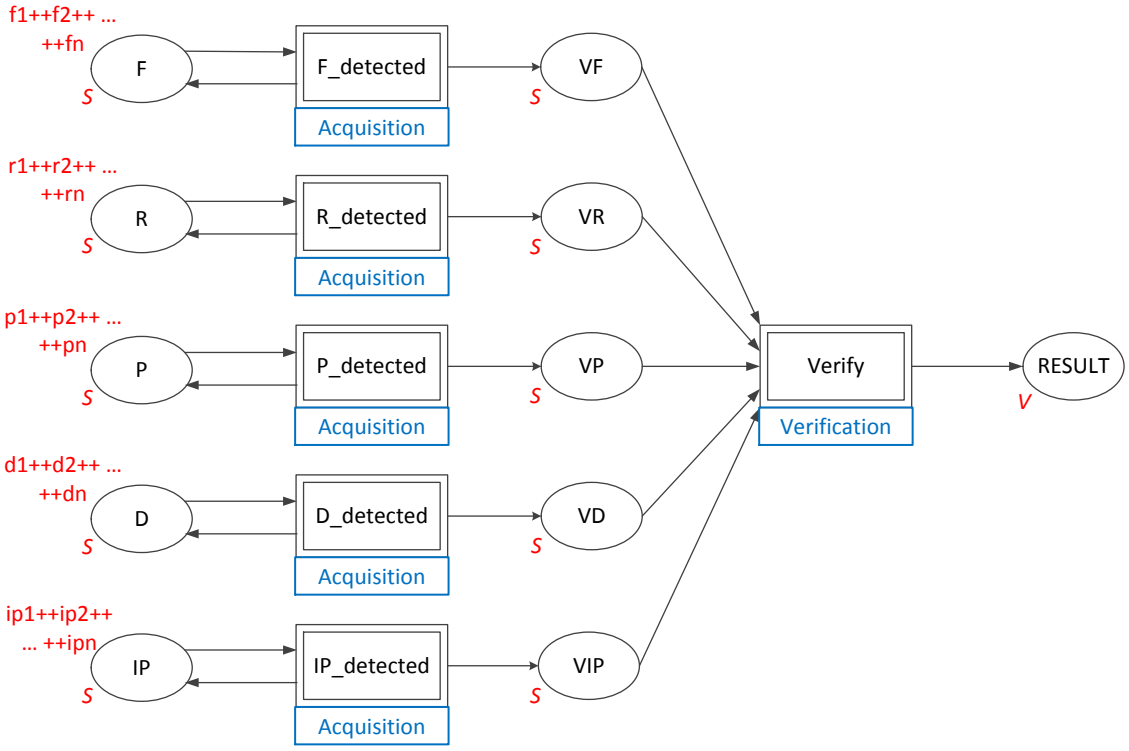
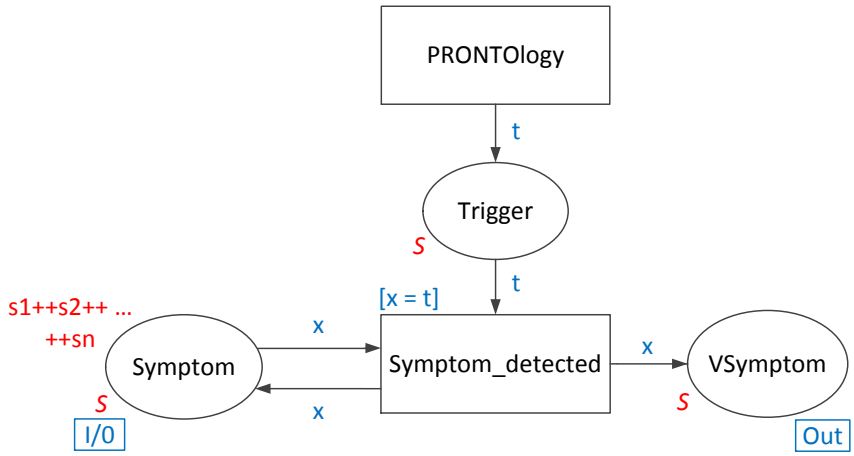


Figure 3. CP-net: Acquisition module.



The CP-net modeling language assumes the application of particular elements presented in Section 3. They have been also indicated in Figure 2, called the primary module. For the purpose of malware modeling, we have defined color types and inscriptions as presented in the following listing:

```
colset S = String;
colset I = Integer;
colset Symptoms = List S;
colset V = Product I * S * Symptoms;
```

Type `S` is a string. In this way, particular assets are described by variables `f`, `r`, `p`, `d`, `ip` of type `S`, e.g., file `C:\[WINDIR]\System32\svchost.exe` or IP address `66.232.126.195`. Type `I` is an integer and is used as threat identity number, and type `Symptoms` is a list of strings describing assets suspected to be infected by malware. Type `V` is a product that indicates the threat identity number, the name of the threat and a list of symptoms that were used for identifying which attack was executed in the system.

In Figure 3, the symptom acquisition process and co-operation with the PRONTOlogy module is presented. The Symptom place is an input/output port that indicates appropriate places F, R, P, D and IP from the higher level module (Figure 2). In the acquisition module, tokens that represent filtered suspicious activities identified by the PRONTOlogy are passed to the VSymptom place if the same token exists at the Symptom place. Identified suspicious actions mark the VSymptom place for further processing by the Verify transition. Marking of the Symptom place contains all tracked elements of the monitored system, e.g., all IP addresses with which the system may communicate and from which it downloads malicious software. Transition Symptom_detected is developed in order to test the existence of the appropriate token in the Symptom place in case the Trigger place is marked. If the compared tokens are different, the transition does not react. The conformity of tokens is required by the guard $[x = t]$. The CP-net model of the acquisition module takes into account the situation when more than one malware type has the same individual symptoms. In this case, marking of the Symptom place is not reduced, while the Symptom_detected transition is enabled.

This module shows an important role of the PRONTOlogy [28,29] in the identification of suspicious events flowing from the system calls' monitor. This identification process is based on the set of rules describing suspicious activities using the same semantics as CP-nets (color sets, places and their appropriate markings), e.g., modification of the registry by a particular process, but with the use of Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL). The rules enable one to recognize single symptoms and are created by the malware analyst after a deep analysis of particular malware behavior. This process is necessary for sifting through thousands of events flowing from the sensor, increasing the efficiency of the PRONTOnet. The rules are generic and enable one to detect obfuscated malware and newly created zero-day attacks that resemble, at some point, the behavior of existing malicious codes. The identified suspicious actions are passed through to the PRONTOnet module, which now performs contextual analysis on the basis of the events' sequences.

The verification module is presented in Figure 4. It must be noted that substitute transition *Verify* in the primary module represents multiple transitions designed for the identification of various malware types. This indicates that characteristic marking of V places causes enabling of the transition appropriate for particular malware. This triggers the verification process. An exemplary virus detection for the chosen marking is shown in Figure 5, which presents the detection of malware called Virut [31].

Figure 4. CP-net: verification module.

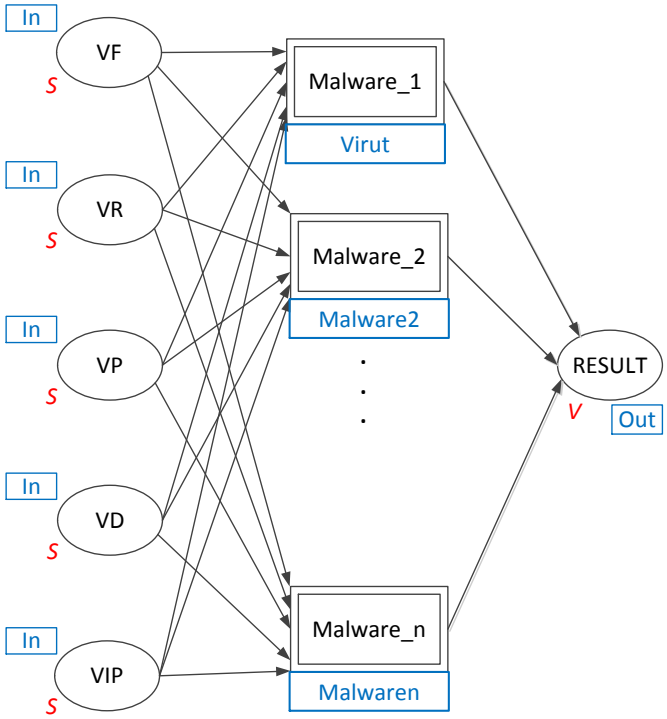
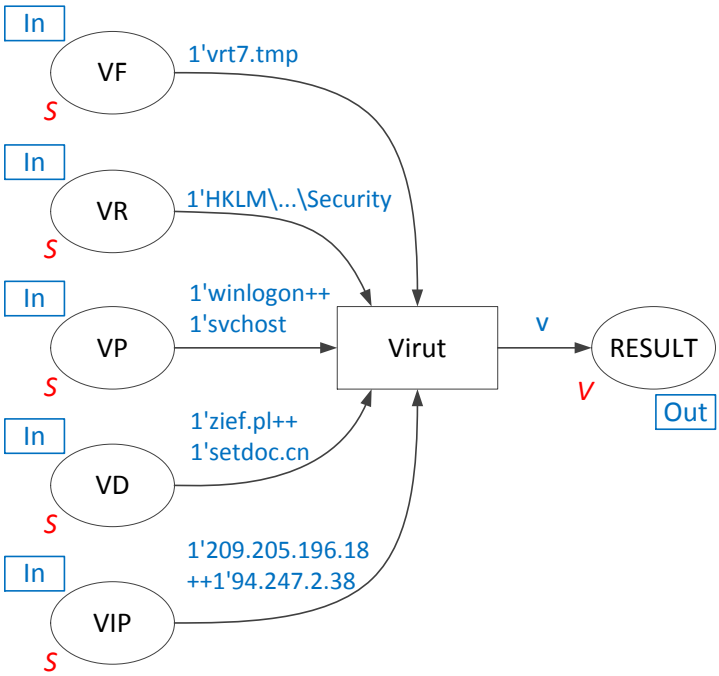


Figure 5. CP-net: Virut module (classifier).



In the presented example, assuming the appearance of the appropriate marking, transition `Virut` is enabled, which, as a consequence, leads to receiving vector `v` informing about detection of the Virut malware. The structure of vector `v` is as follows:

```
1’ 1 | Virut | vrt7.tmp, HKLM\...\Security, winlogon, svchost,
zief.pl, setdoc.cn, 209.205.196.18, 94.247.2.38.
```

The CP-net models presented in this section allow one to easily update the symptoms of a new attack by changing the initial marking of places. Every time a new malware model is entered into the PRONTOnet, the appropriate actualization of places' markings must be performed.

It is also worth emphasizing that the detection of malware is not limited only to the depicted resources. PRONTOnet may be also used for the identification of exploits through the analysis of network traffic and system statistics [32]. If some resource is identified as useful for malware detection, the primary module needs only to be updated with additional places and transitions.

5. Performed Experiments: Verification of the Solution

The presented malware modeling based on CP-nets is to support the detection of malicious software. That is why this chapter presents the verification of the proposed method in terms of the possibility of its application in malicious activity detection and categorization into a specific malware type. The verification process based on a case study has been divided into the following steps:

- Evaluation of the malware modeling process in terms of its applicability and ease of use. An exemplary malware is presented as a CP-net model with the use of software developed for the purpose of verification.
- System data acquisition. The simulated malicious software modifies the system, files, registry and connects to its command-and-control (C&C) servers. Moreover, it executes harmful activities. All of the observed actions, including both legitimate one and those made by malware, have been registered.
- Malware detection experiment. Suspicious activities are assigned to malware CP-net models. The detection mechanism reveals which attack was detected. The vector containing information about detected suspicious activities and their similarity to the modeled malware is presented. For the purpose of this article, we describe one scenario presenting the possible detection of Virut, the virus that infects executable files and opens a back door to the compromised computer [31].

5.1. Evaluation of Cyber Attack CP-net Model Construction

In order to define the requirements for cyber attack models according to [25], the following questions need to be answered: What is the purpose? What do we want to learn about the system by making this kind of model? What kinds of properties are we interested in investigating?

Without initially answering these questions in some detail, it is impossible to make a good model, and we shall be unable to decide what should be included in the model and what can be abstracted away without compromising the correctness of the conclusions that will be drawn from investigating the model. Finding the appropriate abstraction level at different points in the development of the system is one of the arts of modeling. The purpose of the modeling of cyber attacks is to understand their nature and behavior in the infected system. Obfuscation methods [33] allow attackers to bypass signature-based security controls, although their activity in the system still has the same nature. Thus, the model of malware is a reflection of the potential attack that may appear. Secondly, modeling of threats should

allow us to observe the modification of malicious activities and to find new threats, so-called zero-day attacks. The similarity to the model—the detection of particular characteristic actions that have been taken from other existing malware—leads to raising an alarm that suspicious activity is detected. Thirdly, and finally, we need to know the value of the risk that our system might be infected and, in case of a successful attack, what level of damage to the system would be observed. The model must also provide the possibility to track the behavior of malware and, on the basis of the observation of different events, decide about the existence of particular malware in the protected system.

The model was constructed in such a way that it reflects all of the computer assets taking part in malware execution, such as the file system, registry, executed processes, IP addresses and the domain names to which the system connects. They are modeled as places in the CP-net model. Thus, set P is formed with the objects of the protected system involved in the detection of the harmful malware activities:

$P = P_{Process} \cup P_{IP\_address} \cup P_{Domain} \cup P_{File} \cup P_{Registry} \cup P_{Sensor}$

Possible transition set T is composed of any actions realized on the system assets:

$T = \{Execute, Create, Modify, Delete, Close, Run, Terminate, Connect, Query, Download, Read, Open, Other\}$

Color set Σ over places P is to reflect different types of assets, e.g., the address of a web page, the location of files, their handlers after execution, sent data, IP address or the domain name of C&C.

$\Sigma = \Sigma_{Process} \cup \Sigma_{IP\_address} \cup \Sigma_{File} \cup \Sigma_{Domain} \cup \Sigma_{Registry}$

Initial marking M_0 of places P allows one to indicate particular assets that must be tracked by the proposed method in order to reflect different events' characteristics, *i.e.*, a particular file name, the file path, the registry entry, *etc*. In the proposed model, M_0 is as follows:

$M_0 = M_{Process} \cup M_{IP\_address} \cup M_{File} \cup M_{Domain} \cup M_{Registry}$, where:

$M_P = \{svchosts, rundll32, csrsc, ...\}$ indicate particular processes,

$M_{IP\_address} = \{209.205.196.18, 66.232.126.195, 94.247.2.38234, ...\}$, IP addresses,

$M_F = \{vrt7.tmp, ntdll.dll, 8.tmp, 9.tmp, ...\}$, files,

$M_D = \{horobl.cn, setdoc.cn, zief.pl, irc.zief.pl, proxim.ircgalaxy.pl, ...\}$, domains,

$M_R = \{HKLM/.../FirewallPolicy, HKU/.../UpdateHost, ...\}$, registry entries.

Set A is a set of directed arcs that connect places to transitions and transitions to places:
$A \subseteq (P \times T) \cup (T \times P)$. They enable one to reflect the sequence of actions on system assets.

For the purpose of particular malware modeling and storing their specifications, a dedicated application, called the CP-net malware modeling tool, was developed (CPN MM).

Let us focus on the modeling of cyber attacks with the use of the CPN MM tool. As an example, showing the model's expressiveness, the Virut malware [31] has been chosen. The first window of the CPN MM is shown in Figure 6. It presents the list of modeled cyber attacks and Virut malware as the last one. Whenever an expert wants to model a new cyber attack, he needs to choose the "Add" button and fill in data, as presented in Figure 7. Symptoms of particular malware are edited in the window presented in Figure 8. It shows a drag and drop tool, which allows one to add subsequent places to the model easily.

Figure 6. CP-net malware modeling tool: First window.

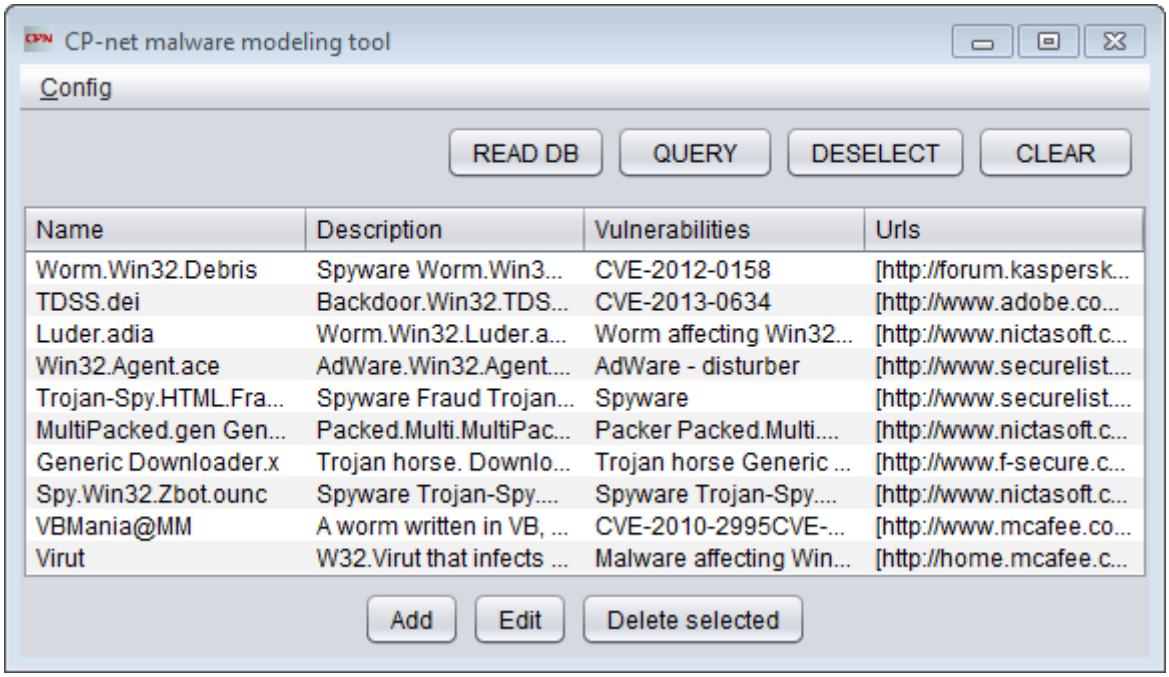


Figure 7. Malware editor window of the CP-net malware modeling tool (CPN MM).

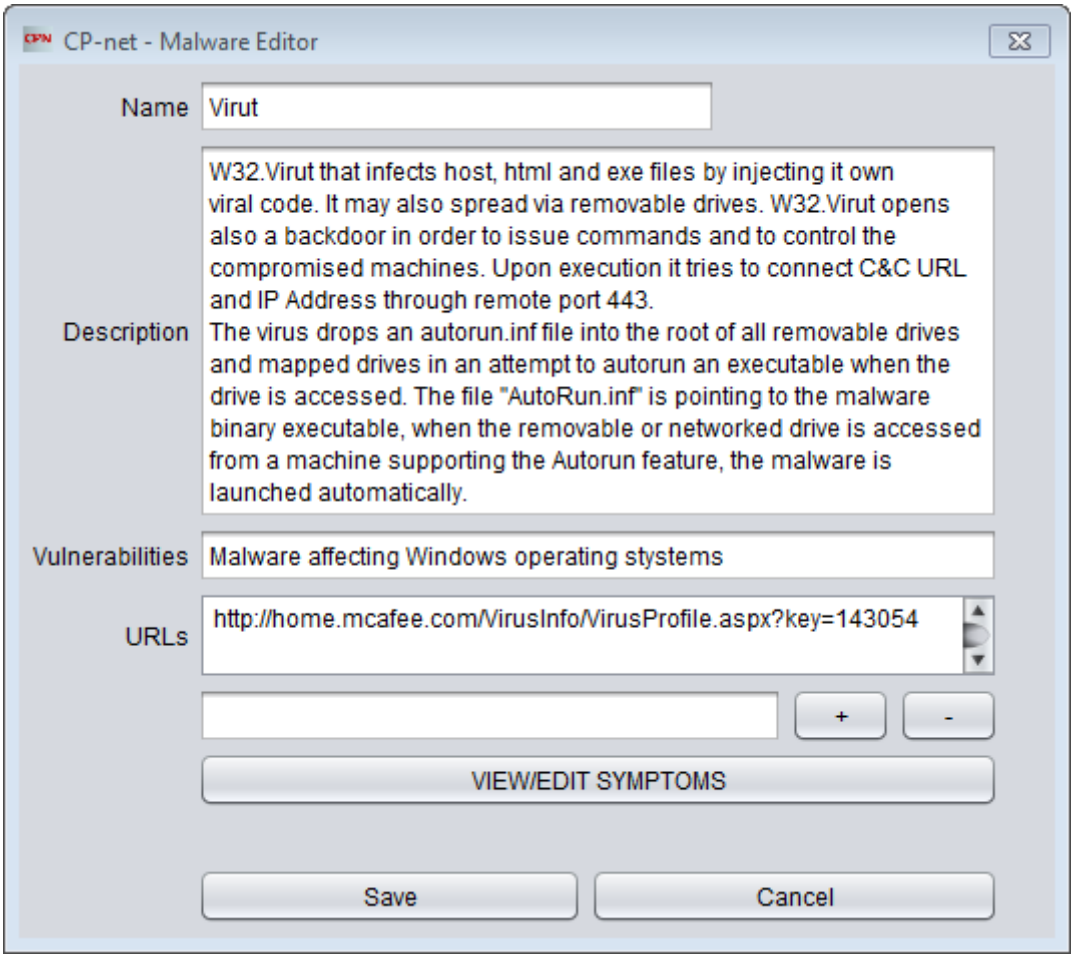
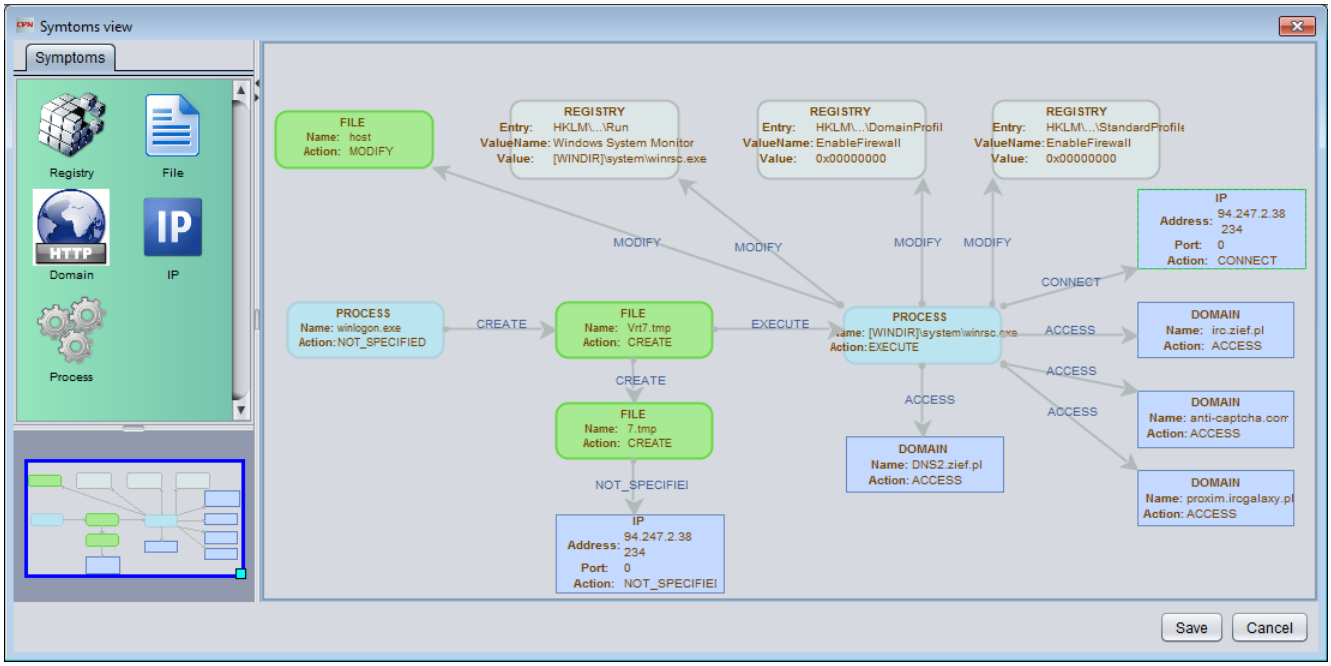


Figure 8. Editor for malware symptoms.



5.2. Data Acquisition

The verification was made with the use of the most popular target of cyber infections, the Microsoft Windows operating system. The authors of this article do not claim that this is the most vulnerable system. In the authors' opinion, the reason for cyber attacks on the Windows operating system is the popularity of the system and the potentially high gain from the conducted attacks. Microsoft products are very popular, which makes them attractive for cyber criminals.

For the observation of activities, applications, services and network connections in the native Microsoft Windows 7 operating system environment, the Sysinternals Suite utility package [34] was used. The Sysinternals Suite is a set of over 70 advanced diagnostic and troubleshooting programs for the Windows platform [35]. The majority of events were observed with the Process Monitor utility [36], a part of the Sysinternals Suite. The Process Monitor is an advanced monitoring application for Windows that registers events that relate to the file system, registry and process activity in real time. It enables monitoring event properties, such as session IDs, user names, process information, thread stacks, simultaneous logging to a file, *etc*. It is a powerful utility that supports the PRONTOlogy module with detailed information on the activities performed in the protected system.

For the purpose of data acquisition, it is also possible to use API hooking tools [37,38]; however, they inject themselves (like viruses) into the processes; thus, they can affect the results of the verification. In the case of the utilization of the PRONTO malware hunting tool for the detection of network attacks, various network utilities, e.g., SNORT, ARAKIS, iptables, should also be used.

Having stored CP-net models of cyber attacks in the database, it is possible to go further with the experiment into the malware detection phase. Generally, the aim of such experiments is not only to identify existing malware that was obfuscated, but also zero-day attacks that have, to some degree, similar behavior to the already identified one. In this article, we present only one scenario with the

detection of the exemplary Virut malware. It is however possible, with an appropriate construction of the verification module, to detect zero-day attacks that are composed of several activities characteristic for already known attacks.

Listing 1. Example of one regular and three suspicious events acquired in the first scenario.

```
<event>
 <ProcessIndex>14340</ProcessIndex>
 <Time_of_Day>17:20:21,1001813</Time_of_Day>
 <Process_Name>WINLOGON.EXE</Process_Name>
 <PID>2728</PID>
 <Operation>ReadFile</Operation>
 <Path>C:\Windows\Temp\vrt7.tmp</Path>
 <Result>SUCCESS</Result>
 <Detail>Offset: 734 720, Length: 16 384, Priority: Normal</Detail>
</event>
<event>
 <ProcessIndex>14560</ProcessIndex>
 <Time_of_Day>17:22:25,1104786</Time_of_Day>
 <Process_Name>ThreatProc.exe</Process_Name>
 <PID>6043</PID>
 <Operation>RegSetValueEx</Operation>
 <Path>HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\
 Windows System Monitor: "C:\Windows\system\winrsc.exe"
 </Path>
 <Result>SUCCESS</Result>
 <Detail>Type: REG_SZ, Length: 24, Data: SimSun-ExtB</Detail>
</event>
<event>
 <ProcessIndex>16640</ProcessIndex>
 <Time_of_Day>17:22:36,2548113</Time_of_Day>
 <Process_Name>WINWORD.EXE</Process_Name>
 <PID>6733</PID>
 <Operation>RegQueryKey</Operation>
 <Path>HKLM</Path>
 <Result>SUCCESS</Result>
 <Detail>Query: HandleTags, HandleTags: 0x0</Detail>
</event>
<event>
 <ProcessIndex>19240</ProcessIndex>
 <Time_of_Day>17:47:02,1294174</Time_of_Day>
 <Process_Name>mmirc.exe</Process_Name>
 <PID>12188</PID>
 <Operation>TCP Connect</Operation>
 <Path>MalwareTest1-VAIO:55052 -> irc.zief.pl:6667</Path>
 <Result>SUCCESS</Result>
 <Event_Class>Network</Event_Class>
 <Image_Path>C:\Windows\Temp\mmirc.exe</Image_Path>
 <Session>1</Session>
</event>
```

5.3. Malware Detection Scenario

Within one minute of Windows 7 OS operation, thousands or even hundreds of thousands of single events may be observed. The report from the Process Monitor includes every action that took place in the system. It includes both regular and suspicious activities. For the purpose of verification and, in particular, the generation of these unwanted activities, different machines were infected by various

malware. At the same time, many legitimate programs were executed on these machines in order to simulate legitimate user activity. This allowed us to generate background regular events. The Virut scenario shows the steps of the PRONTO operation in terms of malware detection on the basis of the CP-net model.

5.3.1. Virut Scenario

The data acquisition phase allows one to gather information about events collected by the Process Monitor. Obviously, the whole file with captured events will not be presented in this chapter, although an exemplary excerpt from it is presented in Listing 1. The events presented in Listing 1 are processed, and XML data is lifted to the semantic metadata [39]. Based on this example, the following instances are inserted into the ontology model (as ABox entries):
for the first event:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#Event_1; an instance of the event class
http://wil.waw.pl/secor/PRONTOlogy.owl#eventName(http://wil.waw.pl/
secor/PRONTOlogy.owl#Event_1, "winlogon_read_vrt.7")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResProcess_2728
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResProcess_2728, "winlogon.exe")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResFile_1
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResFile_1, "vrt7.tmp")
http://wil.waw.pl/secor/PRONTOlogy.owl#read(http://wil.waw.pl/secor/
PRONTOlogy.owl#ResProcess_2728, http://wil.waw.pl/secor/
PRONTOlogy.owl#ResFile_1)
http://wil.waw.pl/secor/PRONTOlogy.owl#hasResource(http://wil.waw.pl/
secor/PRONTOlogy.owl#Event_1, http://wil.waw.pl/secor/PRONTOlogy.owl
#ResProcess_2728)
```

for the second event:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#Event_2; an instance of the event class
http://wil.waw.pl/secor/PRONTOlogy.owl#eventName(http://wil.waw.pl/secor/
PRONTOlogy.owl#Event_2, "ThreadProc_modify_Windows_System_Monitor")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResProcess_6043
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResProcess_6043, "ThreatProc.exe")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResRegistry_1
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResRegistry_1, "HKEY_LOCAL_MACHINE\SOFTWARE\
Microsoft\Windows\CurrentVersion\Run\Windows System Monitor:
C:\Windows\system\winrsc.exe")
http://wil.waw.pl/secor/PRONTOlogy.owl#modify(http://wil.waw.pl/secor/
PRONTOlogy.owl#ResProcess_6043, http://wil.waw.pl/secor/
PRONTOlogy.owl#ResRegistry_1)
http://wil.waw.pl/secor/PRONTOlogy.owl#hasResource(http://wil.waw.pl/
secor/PRONTOlogy.owl#Event_2, http://wil.waw.pl/secor/PRONTOlogy.owl
#ResProcess_6043)
```

for the third event:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#Event_3; an instance of the event class
http://wil.waw.pl/secor/PRONTOlogy.owl#eventName(http://wil.waw.pl/secor/
PRONTOlogy.owl#Event_3, "Winword_read_HKLM")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResProcess_6733
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResProcess_6733, "Winword.exe")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResRegistry_2
```

```
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResRegistry_2, "HKLM")
http://wil.waw.pl/secor/PRONTOlogy.owl#read(http://wil.waw.pl/secor/
PRONTOlogy.owl#ResProcess_6733, http://wil.waw.pl/secor/
PRONTOlogy.owl#ResRegistry_2)
http://wil.waw.pl/secor/PRONTOlogy.owl#hasResource(http://wil.waw.pl/
secor/PRONTOlogy.owl#Event_3, http://wil.waw.pl/secor/PRONTOlogy.owl
#ResProcess_6733)
```

for the fourth event:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#Event_4; an instance of the event class
http://wil.waw.pl/secor/PRONTOlogy.owl#eventName(http://wil.waw.pl/secor/
PRONTOlogy.owl#Event_4, "mmirc_connect_irc_zief_pl")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResProcess_12188
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResProcess_12188, "mmirc.exe")
http://wil.waw.pl/secor/PRONTOlogy.owl#ResDomain_1
http://wil.waw.pl/secor/PRONTOlogy.owl#resourceName(http://wil.waw.pl/
secor/PRONTOlogy.owl#ResDomain_1, "irc.zief.pl")
http://wil.waw.pl/secor/PRONTOlogy.owl#connect(http://wil.waw.pl/secor/
PRONTOlogy.owl#ResProcess_12188, http://wil.waw.pl/secor/
PRONTOlogy.owl#ResDomain_1)
http://wil.waw.pl/secor/PRONTOlogy.owl#hasResource(http://wil.waw.pl/
secor/PRONTOlogy.owl#Event_4, http://wil.waw.pl/secor/PRONTOlogy.owl
#ResProcess_12188)
```

The acquisition module that uses the PRONTOlogy filters out the events gathered with the use of a set of rules [28]. The rules that are valid in the presented scenario allow one to infer that three of the above events are suspicious. On the basis of the rules, the following facts are inferred:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#Place_1; a member of the file class
http://wil.waw.pl/secor/PRONTOlogy.owl#hasPlace(http://wil.waw.pl/secor/
PRONTOlogy.owl#Event_1, http://wil.waw.pl/secor/ PRONTOlogy.owl#Place_1).
http://wil.waw.pl/secor/PRONTOlogy.owl#hasColour(http://wil.waw.pl/secor/
PRONTOlogy.owl#Place_1, http://wil.waw.pl/secor/PRONTOlogy.owl#ResFile_1).

http://wil.waw.pl/secor/PRONTOlogy.owl#Place_2; a member of the registry class
http://wil.waw.pl/secor/PRONTOlogy.owl#hasPlace(http://wil.waw.pl/secor/
PRONTOlogy.owl#Event_2, http://wil.waw.pl/secor/ PRONTOlogy.owl#Place_2).
http://wil.waw.pl/secor/PRONTOlogy.owl#hasColour(http://wil.waw.pl/secor/
PRONTOlogy.owl#Place_2, http://wil.waw.pl/secor/PRONTOlogy.owl#ResRegistry_1).

http://wil.waw.pl/secor/PRONTOlogy.owl#Place_3; a member of the domain class
http://wil.waw.pl/secor/PRONTOlogy.owl#hasPlace(http://wil.waw.pl/secor/
PRONTOlogy.owl#Event_2, http://wil.waw.pl/secor/ PRONTOlogy.owl#Place_2).
http://wil.waw.pl/secor/PRONTOlogy.owl#hasColour(http://wil.waw.pl/secor/
PRONTOlogy.owl#Place_2, http://wil.waw.pl/secor/PRONTOlogy.owl#ResDomain_1).
```

Events 1, 2 and 4 (see Listing 1) have been identified as suspicious, whereas Event 3 as regular system activity.

The PRONTOlogy module passes forward to the PRONTOnet module only information about suspicious events in the form of places and appropriate tokens assigned to them (with the use of the `hasColour` object property). It takes place in the acquisition module, as presented in Figure 3. Then, in the PRONTOnet, these tokens are passed to the verification module where marking M_a of places is:

$M_a = M_{File} \cup M_{Domain} \cup M_{Registry}$, where:

- M_{File}=`{vrt7.tmp}`,

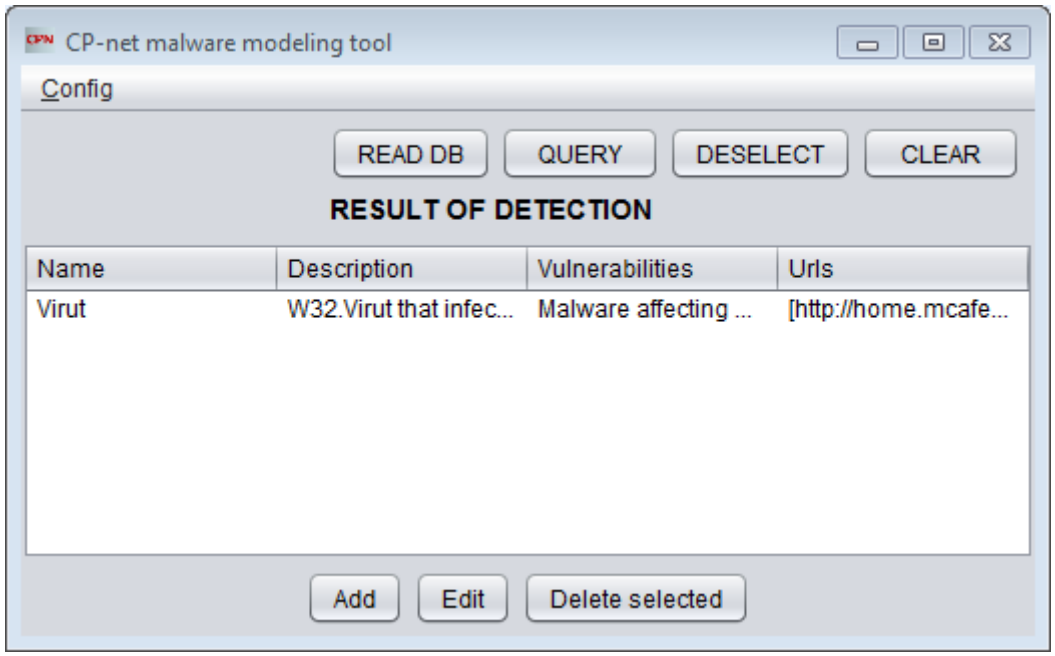
- M_{Domain}={`irc.zief.pl`},
- $M_{Registry}$ = {`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Windows System Monitor: "C:\Windows\system\winrsc.exe"`}.

For the machine described in this scenario, the detection realized with the use of CPN MM and marking M_a allowed for identifying the Virut attack. The result vector is as follows:

```
1' 1 | Virut | vrt7.tmp, irc.zief.pl, Windows System Monitor:
"C:\Windows\system\winrsc.exe"
```

The detection of the Virut malware is shown in Figure 9.

Figure 9. Result of the detection of the Virut malware.



Realization of this scenario allowed us to prove that the ontology model, as well as the applied reasoning rules [40] were successfully adapted to the detection of single malicious incidents. Then, these incidents were collected and compared with the CP-net models created with the use of the CPN MM tool. As a result, the Virut malware has been detected.

The CP-net model described in Section 4 allowed us to reflect all crucial system resources that can be affected by malware. Therefore, it was possible to identify operations performed by an obfuscated code.

6. Conclusions

The article tackles the problem of malware modeling for the purpose of the detection process. It proposes a new approach to behavioral malicious code analysis based on CP-net models in order to inform security stakeholders about suspicious activities observed in the monitored system. This should lead to faster and more appropriate decisions that mitigate the negative results of the conducted cyber attack and, as a consequence, the development of new signatures to avoid similar threats.

The malware behavior model used in the PRONTO hunting tool has been formally specified. It consists of a hierarchical CP-net model allowing for the detection of not only well-known cyber attacks,

but also zero-day attacks constructed from the existing parts of malicious codes. Moreover, the method is resilient to the very popular obfuscation of the malicious code. It precisely detects single or groups of suspicious activities caused by the malware. Our approach is focused both on the formal modeling, as well as the success rate of the detection process. It is also generic in terms of the detection of multiple types of malware in opposition to the methods focused on some specific attacks (e.g., on e-mails, web browsers). The proposed method tracks and traces events related to various system assets, like files, system registry, processes, communicated domains or IP-addresses. This list could be extended if needed.

The limitation of our method can be the lack of the possibility to use it in virtualized machines, so-called sandboxes. The reason is the fact that the vast majority of malware is able to detect the virtualized environment and does not activate itself in such an environment. In order to overcome this inconvenience, we are working on a honeypot that could more easily get infected by malware and enable us to identify suspicious or harmful activities characteristic of this particular malware, as well as to perform an analysis of its code in the controlled environment.

The CP-net-based malware model defines behavioral patterns that express software activities on the operating system. The detection method that uses the model moves through the places in the CP-net on the basis of the interaction with sensors that record sequences of activities and verify their potential harmfulness (the acquisition phase) and their analysis (the detection phase).

It has been shown that the proposed approach to modeling and the resulting CP-net cyber attack models constructed with the use of the CP-net malware modeling tool allow one to identify known malware.

Due to the limited size of this article, it is not possible to report all of the experiments that have been performed; however, it is also possible to detect zero-day attacks, the code of which has been obfuscated. PRONTOnet is able to pass on the alarms about the observed threat and the symptoms that indicate the existence of particular malware.

The proposed method of malware modeling and its detection is planned to be adapted at least in the Federated Cyber Defence System being developed in Poland; however, its applicability is much wider. It can be used in honeypots spread in the monitored system in the form of so-called sandboxes without any or with vulnerable security controls in order to trace and track unwanted activities generated by the software, as well as the users of the system. The advantage of the method should be also noticed by companies utilizing only signature-based anti-virus applications. In particular, PRONTO fulfills the current need to shorten the time of malware detection from the moment it has established itself in the system. As mentioned in the introductory chapter of this article, over 50% of malware was detected after months and about 30% after weeks of infection. Before being identified, cyber attacks can cause irreversible losses and damage in the system. The alarm raised by PRONTO and the in-time introduction of appropriate security measures and malware removal can mitigate the risk of potential losses. Moreover, faster detection of new malware should lead to faster delivery of appropriate signatures constructed on the basis of static code analysis, which is a current need of companies producing anti-virus applications.

Acknowledgments

This work has been partially supported by the Polish National Centre for Research and Development, Project No. PBS1/A3/14/2012 "Sensor data correlation module for detection of unauthorized actions and support of decision process" and by the European Regional Development Fund, the Innovative Economy Operational Programme, under the Intelligent Information System for Global Monitoring, Detection and Threat Identification (INSIGMA) Project No. 01.01.02-00-062/09.

Author Contributions

All authors have contributed to the study and preparation of the article. All authors have read and approved the final manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Gostev, A. Kaspersky Security Bulletin: Statistics 2008. Available online: http://securelist.com/analysis/kaspersky-security-bulletin/36241/ (accessed on 16 December 2014).
2. Funk, C.; Garnaeva, M. Kaspersky Security Bulletin. The Overall Statistics for 2013. Available online: http://securelist.com/analysis/kaspersky-security-bulletin/58265/ (accessed on 16 December 2014).
3. Singh, R.; Singh, P.; Duhan, M. An effective implementation of security based algorithmic approach in mobile adhoc networks. *Hum.-centric Comput. Inf. Sci.* **2014**, *4*, doi:10.1186/s13673-014-0007-9.
4. Takeshi, A.; Masaki, K.; Murakami, T. Cyber Security Trend—Annual Review 2012. Available online: http://www.nri-secure.co.jp/news/2012/pdf/cyber_security_trend_report_en.pdf (accessed on 16 December 2014).
5. Bencsáth, B.; Pék, G.; Buttyán, L.; Félegyházi, M. The Cousins of Stuxnet: Duqu, Flame, and Gauss. *Future Internet* **2012**, *4*, 971–1003.
6. Chen, Y.L.; Yau, H.T.; Yang, G.J. A Maximum Entropy-Based Chaotic Time-Variant Fragile Watermarking Scheme for Image Tampering Detection. *Entropy* **2013**, *15*, 3170–3185.
7. Alsaedi, R.; Constantinescu, N.; Rādulescu, V. Nonlinearities in Elliptic Curve Authentication. *Entropy* **2014**, *16*, 5144–5158.
8. Verizon RISK Team. Verizone 2012 Data Breach Investigations Report. Available online: http://www.verizonenterprise.com/DBIR/2012/ (accessed on 16 December 2014).
9. McAfee and HB Garry Solution Brief. Extend McAfee Total Protection for Endpoint with HBGary Digital DNA and Responder. Available online: http://www.mcafee.com/us/resources/solution-briefs/sb-hbgary.pdf (accessed on 29 August 2013).

10. Tibbs, H.; Ambler-Edwards, S.; Corcoran, M. *The Global Cyber Game: Achieving Strategic Resilience in the Global Knowledge Society*; Defence Academy of The United Kingdom: Shrivenham, UK, 2013.
11. Jingle, I.; Rajsingh, E. ColShield: An effective and collaborative protection shield for the detection and prevention of collaborative flooding of DDoS attacks in wireless mesh networks. *Hum.-centric Comput. Inf. Sci.* **2014**, *4*, doi:10.1186/s13673-014-0008-8.
12. Elshaafi, H.; Botvich, D. Trustworthiness Inference of Multi-tenant Component Services in Service Compositions. *J. Converg.* **2013**, *4*, 31–37.
13. Shahabi, C.; Kim, S.H.; Nocera, L.; Constantinou, G.; Lu, Y.; Cai, Y.; Medioni, G.; Nevatia, R.; Banaei-Kashani, F. Janus-Multi Source Event Detection and Collection System for Effective Surveillance of Criminal Activity. *J. Inf. Process. Syst.* **2014**, *10*, 1–22.
14. Adair, S.; Deibert, R.; Rohozinski, R.; Villeneuve, N.; Walton, G. Shadows in the Cloud: Investigating Cyber Espionage 2.0. Available online: https://www.f-secure.com/weblog/archives/Shadows_In_The_Cloud.pdf (accessed on 16 December 2014).
15. Lee, J.D.; Sin, C.H.; Park, J.H. PPS-RTBF: Privacy Protection System for Right To Be Forgotten. *J. Converg.* **2014**, *5*, 37–40.
16. Juneja, M.; Sandhu, P.S. A New Approach for Information Security using an Improved Steganography Technique. *J. Inf. Process. Syst.* **2013**, *9*, 405–424.
17. Szwed, P.; Skrzyński, P. A new lightweight method for security risk assessment based on fuzzy cognitive maps. *Int. J. Appl. Math. Comput. Sci.* **2014**, *24*, 213–225.
18. Ihm, H. Mining Consumer Attitude and Behavior. *J. Converg.* **2013**, *4*, 29–35.
19. Canali, D.; Lanzi, A.; Balzarotti, D.; Kruegel, C.; Christodorescu, M.; Kirda, E. A Quantitative Study of Accuracy in System Call-based Malware Detection. In Proceedings of the 2012 International Symposium on Software Testing and Analysis, Minneapolis, MN, USA, 16–20 July 2012; pp. 122–132.
20. Lanzi, A.; Balzarotti, D.; Kruegel, C.; Christodorescu, M.; Kirda, E. AccessMiner: Using System-centric Models for Malware Protection. In Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 4–8 October 2010; pp. 399–412.
21. Egele, M.; Kruegel, C.; Kirda, E.; Yin, H. Dynamic Spyware Analysis. In Proceedings of the 2007 Usenix Annual Conference (Usenix '07), Santa Clara, CA, USA, 17–22 June 2007; pp. 233–246.
22. Fredrikson, M.; Jha, S.; Christodorescu, M.; Sailer, R.; Yan, X. Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors. In Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP '10), Oakland, CA, USA, 16–19 May 2010; pp. 45–60.
23. Chandramohan, M.; Tan, H.B.K.; Briand, L.; Shar, L.K.; Padmanabhuni, B. A scalable approach for malware detection through bounded feature space behavior modeling. In Proceedings of 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), Palo Alto, CA, USA, 11–15 November 2013; pp. 312–322.
24. Wen, S.; Zhou, W.; Zhang, J.; Xiang, Y.; Zhou, W.; Jia, W.; Zou, C. Modeling and Analysis on the Propagation Dynamics of Modern Email Malware. *IEEE Trans. Dependable Secur. Comput.* **2014**, *11*, 361–374.

25. Jensen, K.; Kristensen, L. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, 1st ed.; Springer-Verlag: Berlin/Heidelberg, Germany, 2009.
26. Szpyrka, M. Analysis of VME-Bus communication protocol—RTCP-net approach. *Real-Time Syst.* **2007**, *35*, 91–108.
27. Petri, C.A. *Communication with Automata*; Technical report, RADC-TR-65-377; Griffiss Air Force Base: New York, NY, USA, 1966.
28. Jasiul, B.; Śliwa, J.; Gleba, K.; Szpyrka, M. Identification of malware activities with rules. In Proceedings of the Federated Conference on Computer Science and Information Systems, Warsaw, Poland, 7–10 September 2014.
29. Jasiul, B.; Szpyrka, M.; Śliwa, J. Malware behavior modeling with Colored Petri nets. In *Computer Information Systems and Industrial Management*, Proceedings of the 13th IFIP TC8 International Conference, CISIM 2014, Ho Chi Minh City, Vietnam, 5–7 November 2014; Saeed, K., Snášel, V., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Lecture Notes in Computer Science, Volume 8838; pp. 667–679.
30. Śliwa, J.; Gleba, K.; Chmiel, W.; Szwed, P.; Głowacz, A. IOEM-Ontology Engineering Methodology for Large Systems. In *Computational Collective Intelligence. Technologies and Applications*, Proceedings of the Third International Conference, ICCCI 2011, Gdynia, Poland, 21–23 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; Lecture Notes in Computer Science, Volume 6922; pp. 602–611.
31. Virus Profile: W32/Virut.n.gen Characteristics by McAffee. Available online: http://home.mcafee.com/virusinfo/virusprofile.aspx?key=154055 (accessed on 16 December 2014).
32. Bereziński, P.; Szpyrka, M.; Jasiul, B.; Mazur, M. Network anomaly detection using parameterized entropy. In *Computer Information Systems and Industrial Management*, Proceedings of the 13th IFIP TC8 International Conference, CISIM 2014, Ho Chi Minh City, Vietnam, 5–7 November 2014; Saeed, K., Snášel, V., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Lecture Notes in Computer Science, Volume 8838; pp. 465–478.
33. Uto, N. A Methodology for Retrieving Information from Malware Encrypted Output Files: Brazilian Case Studies. *Future Internet* **2013**, *5*, 140–167.
34. Russinovich, M.; Margosis, A. *Windows Sysinternals Administrator's Reference*; Microsoft Press: Redmond, WA, USA, 2011.
35. Microsoft Technet—Sysinternals. Available online: http://technet.microsoft.com/en-us/sysinternals/ (accessed on 16 December 2014).
36. Russinovich, M.; Cogswell, B. *Process Monitor*. Available online: http://technet.microsoft.com/en-us/sysinternals/bb896645/ (accessed on 16 December 2014).
37. API hooking revealed. Available online: http://www.codeproject.com/Articles/2082/API-hooking-revealed (accessed on 16 December 2014).
38. EasyHook. Available online: http://easyhook.codeplex.com/ (accessed on 16 December 2014).
39. Elsayed, E.; Eldahshan, K.; Tawfeek, S. Automatic evaluation technique for certain types of open questions in semantic learning systems. *Hum.-centric Comput. Inf. Sci.* **2013**, *3*, doi:10.1186/2192-1962-3-19.

40. Verma, O.; Jain, V.; Gumber, R. Simple Fuzzy Rule Based Edge Detection. *J. Inf. Process. Syst.* **2013**, *9*, 575–591.