Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 101–110
DOI: 10.15439/2014F265
ACSIS, Vol. 2

Identification of malware activities with rules

Bartosz Jasiul, Joanna Śliwa, Kamil Gleba
Military Communication Institute,
C4I Systems' Department,
ul. Warszawska 22a, 05-130 Zegrze, Poland
Email: {b.jasiul, j.sliwa, k.gleba}@wil.waw.pl

Marcin Szpyrka
AGH University of Science and Technology,
Department of Applied Computer Science,
al. Mickiewicza 30, 30-059 Kraków, Poland
Email: mszpyrka@agh.edu.pl

***Abstract*—The article describes the method of malware activities identification using ontology and rules. The method supports detection of malware at host level by observing its behavior. It sifts through hundred thousands of regular events and allows to identify suspicious ones. They are then passed on to the second building block responsible for malware tracking and matching stored models with observed malicious actions. The presented method was implemented and verified in the infected computer environment. As opposed to signature-based antivirus mechanisms it allows to detect malware the code of which has been obfuscated.**

I. Introduction

OVERWHELMING number of computer systems are connected to each other by global network – Internet, which allows to produce results beyond those achievable by the individual systems alone. Outcomes of cooperative work and accessibility of information are perceived and appreciated probably by all its users.

The advantages of this technology are available, unfortunately, also for hostile goals. The number of cyber threats arises rapidly from 23 680 646 in 2008 [1] to 1 595 587 670 in 2012 [2], and this is nowadays one of the most vexing problems in computer system security [3]. At the end of 2012 Kaspersky Lab, the Russian producer of antivirus software, reported that [4] *it currently detects and blocks more than 200 000 new malicious programs every day, a significant increase from the first half of 2012, when 125 000 malicious programs were detected and blocked each day on average*.

Although awareness about necessary security appliances seems to be common and the tools used for that purpose are getting more and more advanced, the number of successful attacks targeted on computer systems is growing [5]. They are mostly related to denial of offered services, gaining access or stealing private data, financial fraud, etc. Moreover, the evolution towards cloud computing, increasing use of social networks, mobile and peer-to-peer networking technologies that are intrinsic part of our life today, carrying many conveniences within our personal life, business and government, gives the possibility to use them as tools for cyber criminals and potential path of malware propagation [6]. Computer systems are prone to cyber attacks even though a number of security controls are already deployed [7], [8]. Cyber criminals are focused on finding a way to bypass security controls and gain access into the protected network. For that reason organizations, companies, governments and institutions as well as ordinary citizens all over the world are interested in detection of all attempts of malicious actions targeted on their computer networks and single machines [9].

Malicious activity detection starts with application of various techniques, the success rate of which depends on the reliability of the malware model. Usually they are based on code signatures. Security controls (e.g. antivirus tools) might be maladjusted because signatures of new threats are not identified yet. Hackers often use existing parts of code in order to implement new types of malware. This allows, in return, to quickly develop signatures of new dangerous software. Therefore, the more signatures are deployed the more malicious codes are identified. On the other hand, one of the methods for misleading signature-based detection systems is code obfuscation, the aim of which is generating – from already existing code – a new application that cannot be assessed yet as risky by security controls [10]. This technique is simple to be used and potentially successful. One of the countermeasures in this case is to follow behaviors of malicious software in order to identify them and eliminate from the protected system.

According to the study conducted in 2012 by the Verizon RISK Team [11] with cooperation from many national federal organizations, including e.g. Australian Federal Police, Irish Reporting and Information Security Service, and United States Secret Service *new techniques that speed up the process of malware detection to hours* are necessary. Authors of the report [12] indicate that *antivirus products should be supported by malware behavioral analysis tools in order to detect those of attacks for which signatures were not established*. An existing example of appliance that uses behavioral analysis for advanced persistent threats detection is Digital DNA by HBGary that extends the capabilities of McAfee Total Protection antivirus [13]. Detailed technical specifications of this solution have not been released for public. The product brochure explains that *multiple low level behaviors are identified for every running program or binary*. This leads to conclusion that each application is observed from behavioral perspective. McAfee is proud that the solution allowed to detect last year more 0-day

Work has been partially financed byby the 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 the European Regional Development Fund the Innovative Economy Operational Programme, INSIGMA project no. 01.01.02-00-062/09.

attacks than during the previous five years combined. This indicates the scale of new malware development and efficacy of the behavioral approach.

II. State of the art in malware detection

Currently there are two major techniques seen as prospective for malicious threats detection. One of them is *machine learning* [14] which allows to detect anomalies in the use of host machines by malicious software. This approach is only applicable in systems, for which a model of normal behavior can be established. It is only possible in such an environment where patterns for host machine activity can be identified, e.g. in production environment, SCADA systems, etc. Current usage of computer systems, mobility of users, enormous number of executed applications and visited internet sites cause that setting up a normal behavior model for malware detection is almost impossible. Such a method may also generate too many *false positive* alarms.

Other methods used for identification of malicious actions are based on different forms of specially prepared behavioral patterns prepared in the process of static code analysis or various host-based honeypots and sandboxes. Those patterns can be then applied in detection tools, e.g. rule based engines and complex event processing (CEP) tools. This article presents the method that uses rules [15] in order to identify malicious actions of the host machine and filters out from the number of system activities only these that are typical for malware.

III. Idea of the solution

In our work we proposed and developed behavior-oriented malware hunting tool, so called `PRONTO`, that could be used in parallel to existing signature-based tools.

The main assumption for the introduced method is that the malware was not recognized yet by the signature mechanisms. The aim therefore is to track its suspicious activities in order to find it while running in the system.

`PRONTO` hunting tool performs its activity in two stages (Fig. 1):

- **Filtering of the system events** registered by the system monitors (sensors) to discover the main features of the hostile activity. These features are related to particular objects and actions triggered on that objects – e.g. registry (add entry, modify entry, delete registry entry, etc.), process (start, stop process, etc.), file (copy, delete, run, open, close file, etc.), domain (connect to, etc.), IP address (connect to, etc.);
- **Tracking suspicious activity** in order to discover malicious exploits running in the system. Filtered events are correlated in order to find similarities with the stored malware activities modeled in the form of Colored Petri nets [16]. The result of malware tracking is the alarm that contains information vector about malicious activity, similarity to the known attacks and list of incidents that affected the system.

This article presents only the first stage which is related to capturing events from sensors and analyzing them with an expert system that uses – defined for the purpose of the method – comprehensive ontology, so called `PRONTOlogy`. Registered events in the form of XML objects are sent to the PRONTOntology engine and lifted to add entries to the Knowledge Base. `PRONTOlogy` describes events registered by system monitors and is able, on the basis of rule engine and inference, with the use of specially defined rules [17], to classify an event as potentially suspicious, malicious or regular. As a result, markings of the modeled malware in the form of CP-nets [18][19] are delivered for further analysis.

The second element of the threats' tracking component of the solution is `PRONTOnet` [20]. It provides formal model of malware behavior and allows to track suspicious activities potentially assigning them to the class of known malware types or identifying unknown ones. Known exploits can be invisible to signature-based malware detecting tools after their code has been obfuscated, although their activities can be easily observed. It also happens often that a new malware piece of software is composed of known components from other ones. This results in another behavior pattern that can be tracked as a new exploit, not identified yet. The result of threats tracking stage is an alert informing about identification of suspicious or malicious events with a certain similarity rate to the known malware types.

IV. Identification of malicious actions

Static analysis of malicious code or intelligent algorithms for malware behavior recognition provide patterns that can be defined in low level programming language (e.g. Assembler) or can be represented on the level of operating system activities. In case of our solution the second approach was selected, which means that identification of malicious actions is performed while monitoring actions of the host machine. It was mainly due to availability of tools for operating system monitoring and easiness of processing.

The two-stage malware hunting process presented in this article starts with sifting through a great number of actions that are generated by the up and running operating system. This aimes for identification of those events that should be perceived as suspicious and processed further on. This process should enable automatic filtering of events on the basis of their characteristic features. However, it is not trivial to assume an action is suspicious since the mechanism must catch the context of its invocation in order to assess if it is a regular operating system or user activity, or anomaly that should be investigated further on. For this reason, it was necessary to use a method that could provide the possibility to deduce from the gathered data and analyze possible correlation among events. These requirements were met by the semantic techniques based on ontology and rules that enable to create knowledge base and infer additional facts automatically.

According to [21] *An ontology is an explicit and formal specification of a conceptualization*.

In general, ontology describes a domain of discourse formally. Typically, ontology consists of a finite list of terms, and relationships between those terms. This set describes so

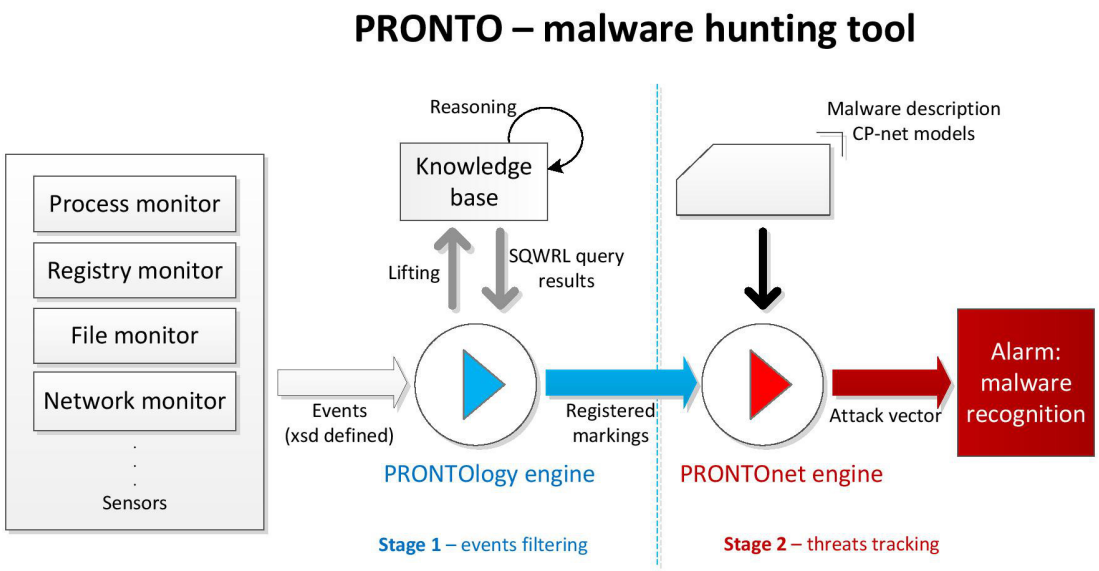

Fig. 1. PRONTO – malware hunting tool

called TBox statements, which are Terminological statements describing the domain in terms of controlled vocabularies. They describe important concepts (classes of objects) of the domain and their properties.

For the purpose of this solution there has been proposed an ontology modeled in the Web Ontology Language (OWL) titled `PRONTOlogy.owl` that describes basic classes and relationships among them. Since the investigated domain needs the description that would enable to reflect and represent facts that a resource executes an action on another resource particular object properties are used. They indicate actions executed on resources and enable to define appropriate triples (e.g. `run (x,y)`, where `x`, `y` are members of `Resource` class and `run` is *object property*, with *domain* and *range* equal to Resource).

Based on TBoxes there can be defined e.g. the following general statements:

```
Event(x)
Resource(y)
ResFile(z)
hasResource(x,y)
Resource(y) is a ResProcess
ResFile(z) is a Resource
run (y,z)
```

where:

- `Event, Resource, ResProcess, ResFile` – are classes,
- `hasResource, run` – are object properties,
- `is a` – is subclass relationship.

According to Fig. 2 the model ontology consists of the three main *classes*: `Event`, `Place`, `Resource`.

In order to differentiate types of resources that perform actions observed by system monitors, there have been defined

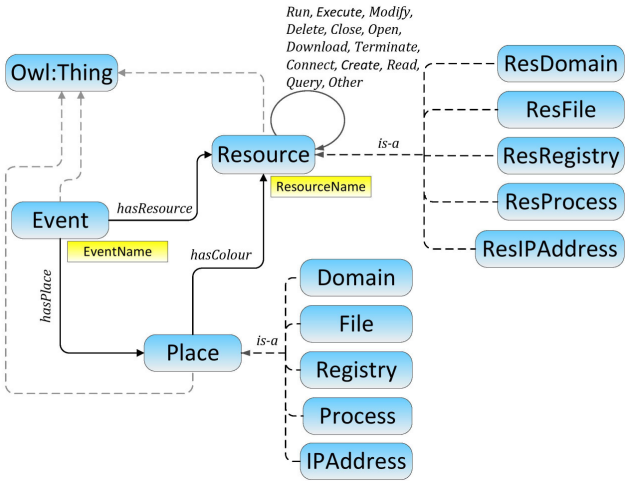

Fig. 2. PRONTOlogy model

the following *subclasses* of the `Resource` class:

- `ResFile` – where the resource is a file,
- `ResRegistry` – where the resource is a registry,
- `ResProcess` – where the resource is a process,
- `ResDomain` – where the resource is a domain the system is trying to connect to,
- `ResIPAddress` – where the resource is an IP address the system is trying to connect to.

In order to indicate particular registry entries, file names, etc. *datatype properties* have been proposed. They describe name of the Resource (`ResourceName`) and name of particular Event (`EventName`).

In order to reflect activities on system resources there have been modeled the following *object properties* that resemble

types of system activities:

- `run` – e.g. running a service, process;
- `open` – e.g. opening a file, registry;
- `close` – e.g. closing a file, application, process;
- `modify` – e.g. modification of registry entry, file, process;
- `execute` – e.g. executing an application;
- `terminate` – e.g. terminating a process;
- `connect` – e.g. connecting to the IP address or domain;
- `query` – e.g. querying the registry entry state;
- `download` – downloading data from remote location;
- `create` – creating a new object, e.g. registry entry, file;
- `delete` – deleting an object, e.g. file, registry entry.

For all *object properties* listed above `Domain` and `Range` are equal to `Resource`, which means that one resource can execute actions on other resources.

There are also additional *object properties* that can reflect the fact:

1) that particular event should be perceived as a `Place`: `hasPlace`, where `Domain = Event, Range = Place`,
2) that particular marking appears for particular `Place`: `hasColor`, where `Domain = Place, Range = Resource`, and
3) that particular event is related to a `Resource`: `hasResource`, where `Domain = Event, Range = Resource`.

The above defined types of *classes* and *object properties* enable to describe events that are registered by system monitors (sensors). Additionally, the model was constructed in such a way that it can reflect the fact that particular observed activity should be perceived as a token in the Petri net - used further on at the second stage of malware hunting tool operation. This process is performed automatically with the use of reasoning rules modelled with Semantic Web Rule Language (SWRL) [22], which offers appropriate expressiveness and tool support in order to use it for the assumed purpose [23].

A. PRONTOlogy engine

The ontology model presented above is used in the PRONTOlogy engine. System activities that are logged by different system sensors form a stream of hundreds to even hundred thousands of events per minute. They record activities of the user and related background activity of the system. In terms of presented solution sensors that cover the spectrum of incidents describing the behavior of different malware types are process, registry, file and network monitors, reflected in the ontology.

Sensors allow to log file system, registry and process/thread activities in real-time. After a proper configuration they enable sifting through incoming events and comprehensive event properties such as session ID numbers, user names, reliable process information, full thread stacks with integrated symbol support for each operation, simultaneous logging to a file, etc.

The first stage of the PRONTOlogy engine operation is devoted to the analysis of this events' stream and classification of single events as either suspicious or regular ones. This classification is assumed as a background activity for the monitoring of the system state realized by the second component of PRONTO, that is threats' tracking (PRONTOnet).

Particular types of malware perform distinctive activities. Each of them is different or, what is more, they can have their types. This entails various malware realization. If malware signature is unknown (the code has been obfuscated), identification of its activity can be done by analyzing system events. The first stage of this process is related to filtering of events and classifying them as neutral or suspicious. In the latter case information about the event is passed down to the next stage – threat tracking.

The stage of events filtering is based on the ontology engine that automatically, with the use of the knowledge base and a set of rules, defines if the registered event is suspicious and should be tracked further by the PRONTOnet module. Knowledge base is created with lifting the information about events registered by sensors to create assertions and facts (entering ABox statements into the knowledge base). Suspicious actions are modeled as instances of the `Place` class. As already mentioned, the rules will provide the possibility to infer facts that particular event indicates existence of a `Place` in the CP-net model (`hasPlace` object property) and therefore particular token (`hasColor` object property) exists and this fact should be passed further on to the PRONTOnet. For instance, the rules can infer that e.g. an event called *winlogon.exe_run_VRT7.tmp* which means that the `winlogon.exe` process has run `VRT7.tmp` file is suspicious and sends on this information to the threat tracing module for further investigation.

As events from sensors are delivered to PRONTOlogy engine, new facts are inserted into the knowledge base in the form of ABox statements. In order to provide additional facts to the knowledge base in terms of appearance of a new token in the CP-net model of particular attack, the rules are proposed. The following listing shows the rule which head defines a condition: an event where a process named *csrss* opens a file named *open.exe*, which in fact is an infected file. When this condition is met, it results in identification of a new `Place` in the CP-net model, which is a `File` with token `open.exe`.

```
Place(?c) ^ Resource(?y) ^ resourceName(?y,
"csrss.exe") ^ open(?y, ?z)   ^ ResFile(?z)
^ resourceName(?z, "open.exe") ->  File(?c)
^ hasColour(?c, ?z)
```

A new set of rules will be prepared whenever new threats appear in the process of system vulnerabilities analysis. For the purpose of the solution verification there has been shown an exemplary set that will provide the possibility to discover markings of places defined in the CP-net model and used in `PRONTOnet`.

V. PRONTOLOGY.OWL EVALUATION

This section is devoted to ontology evaluation which, according to [24], should consist in *validation and verification*

of an ontology in terms of its scope, consistency and expressiveness.

Semantic model defined in PRONTOlogy is devoted to reflect events that occur in the monitored system and enable to identify these suspicious ones. This model has direct relationship with the CP-net through the use of the `Place` class the instances of which are passed over to the PRONTOnet. PRONTOlogy defines:

- The event instance modeled with the use of `Event` class. Data about occurred events is received from the sensors, then lifted to the ontology model as instances of the `Event` class. Each instance has `eventName` (*data property*) defined by the sensor. The description of an event is modeled with the use of `hasResource` *object property* which indicates initiator of the event which is some system resource.
- System resources that are under monitoring by sensors – File, Registry, Process, Domain, IPaddress. They are modeled by the following classes: `ResFile`, `ResRegistry`, `ResProcess`, `ResDomain`, `ResIPAddress`, which are subclasses of the `Resource` class.
- The event description modeled with the use of *object properties* (`run`, `create`, `modify`, `delete`, `download`, `open`, `close`, `read`, `execute`, `terminate`, `connect`, `query`). These *object properties* `domain` and `range` is the `Resource` class, which means that resources perform actions on other resources.
- An abstract `Place` class that defines the fact that particular event is suspicious and should be handled over by the CP-net model for further investigation. This class has five subclasses that define the type of a Place, which in turn results from event originator and reflects Places in the CP-net model.

With the use of the proposed ontology it is possible to describe the event of running a file by particular process. For instance occurrence of `winlogon.exe_run_VRT7.tmp` event would cause inserting of the following instances into the knowledge base:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#
Event_1
http://wil.waw.pl/secor/PRONTOlogy.owl#
eventName(http://wil.waw.pl/secor/
PRONTOlogy.owl#Event_1,
"winlogon.exe_run_VRT7.tmp")
http://wil.waw.pl/secor/PRONTOlogy.owl#
ResProcess_8
http://wil.waw.pl/secor/PRONTOlogy.owl#
resourceName(http://wil.waw.pl/secor/
PRONTOlogy.owl#ResProcess_8,
"winlogon.exe")
http://wil.waw.pl/secor/PRONTOlogy.owl#
ResFile_9
http://wil.waw.pl/secor/PRONTOlogy.owl#
resourceName(http://wil.waw.pl/secor/
PRONTOlogy.owl#ResFile_9, "vrt7.tmp")
http://wil.waw.pl/secor/PRONTOlogy.owl#
run(http://wil.waw.pl/secor/
PRONTOlogy.owl#ResProcess_8,
http://wil.waw.pl/secor/
PRONTOlogy.owl#ResFile_9)
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_8)
```

The stage of events filtering is based on the ontology engine that automatically, with the use of the knowledge base and a set of rules defines if the registered event is suspicious and should be tracked further by the threat tracing module. As already mentioned knowledge base is created with lifting the information about events registered by sensors to create assertions and facts. Suspicious actions are modeled as instances of the `Place` class. The knowledge about the object which is also an instance of the `Place` class is derived by the set of rules proposed for the purpose of PRONTOlogy.

If the following rule is applied:

```
Place(?c)^Resource(?y)^resourceName
(?y, "winlogon.exe")^run(?y, ?z)
^ResFile(?z)^resourceName(?z,
"vrt7.tmp") -> File(?c)^hasColour(?c, ?z)
```

the following instances are added to the knowledge base:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#
Place_1-is a member of File class
(inferred knowledge)
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_9).
```

If the following rule is applied:

```
Event(?e)^Place(?c)^hasResource(?e,?y)
^Resource(?y)^resourceName(?y,
"winlogon.exe")^run(?y, ?z)^ResFile(?z)
^resourceName(?z,  "vrt7.tmp")
-> hasPlace(?e,?c)^File(?c)^
hasColour(?c, ?z)
```

additionally the relation

```
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).
```

is added.

If an event has `hasPlace` relation to any `Place` instance, it is a suspicious event.

The PRONTOlogy defines all entities that are necessary to describe events monitoring system behavior and identify suspicious events. Moreover, the direct relation between ontology and CP-nets has been modeled with the use of the `Place` class. That is why, they satisfy the required scope and expressiveness of ontology.

The second ontology evaluation step consists in checking the ontology consistency. According to [25] *ontology is consistent (also called satisfiable) when it does not contain a contradiction*. The lack of contradiction can be defined in either semantic or syntactic terms. The syntactic definition states that a theory is consistent if there is no such P formula that both P and its negation are provable from the axioms of the theory under its associated deductive system. The ontology model that contains formal definitions of classes, properties and individuals allows inferring new knowledge from knowledge that is already present. The fact that it is based on formal description logic makes it prone to logical reasoning and enables to infer knowledge from existing *facts* and *axioms* [26].

The consistency of `PRONTOlogy.owl` has been verified in the Protégé ontology editing tool (version 3.4.6) [27] using the Pellet 1.5.2 [28] reasoner on a machine with the following configuration:

- Processor: Intel Core i7 (2 cores 2,8 GHz each);
- RAM: 6 GB;
- Operating System: Windows 7 (64 bit).

The consistency check on this machine was successful and `PRONTOlogy.owl` has been proven consistent in 0,022 seconds.

To satisfy verification of events filtering with ontology and reasoning a web service `PRONTOlogyInterface` was implemented in Java programming environment. The service was developed with utilization of Protégé, Pellet, SWRL Jess bridge, and Jess71p2 programming libraries. Web Service was run on the GlassFish Server 3.1.2.

`PRONTOlogyInterface` consists of two programming packages:

- `wil.waw.pl.protegeclass.prontology`
- `wil.waw.pl.prontology`.

Java classes of `wil.waw.pl.protegclass.prontology` package were developed with *Generate Protégé-OWL Java Code* plug-in of Protégé editor. The generator allowed to define Java classes on the basis of `PRONTOlogy.owl` automatically.

Package `wil.waw.pl.prontology` consists of the following classes:

- `InferenceResult.java`,
- `OperationType.java`,
- `ResourceType.java`,
- `PRONTOlogy.java`.

`InferenceResult.java` class defines result code of `PRONTOlogyInterface` service. `OperationType.java` defines types of operations on resources:

```
public enum OperationType {RUN,EXECUTE,
CREATE,MODIFY,DELETE,CLOSE,OPEN,
DOWNLOAD,CONNECT ACCESS,TERMINATE,
QUERY,READ,OTHER}.
```

Enumerate class `ResourceType.java` defines types of resource:

```
public enum ResourceType{RESDOMAIN,
RESIPADDRES,RESUSER,RESREGISTRY,
RESFILE,RESPROCESS}.
```

`PRONTOlogy.java` is the main class of `PRONTOlogyInterface` and it implements the following web methods:

- `readOntologyFromFile()` that reads the ontology from the file;
- `inferKnowledge()` that realizes ontology reasoning;
- `queryForPetriPlace()` that identifies the *Place* in CP-net on the basis of defined knowledge base;
- `queryForPetriToken()` that identifies, on the basis of the knowledge base, a token assigned to a particular *Place*.

VI. Cyber attacks detection – an experiment

A. Data acquisition

The verification based on experimental data was made with the use of the most popular target of cyber infections – Microsoft Windows operating system. The authors do not claim that this is the most vulnerable system. In the authors' opinion the reason of cyber attacks on Windows operating system is the popularity of the system and potentially high gain from 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 Sysinternals Suite utility package [29] was used. The Sysinternals Suite is a set of over 70 advanced diagnostic and troubleshooting programs for the Windows platform. These programs are available for free download from Microsoft's Technet web page [30].

Majority of events were observed with Process Monitor utility [31] – part of the Sysinternals Suite. Process Monitor is an advanced monitoring application for Windows that registers events which relate to 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 PRONTOlogy module with detailed information on activities in the protected system. An example of a single event acquired with Process Monitor is presented in following listing:

```
<event>
<ProcessIndex>14340</ProcessIndex>
<Time_of_Day>17:22:25,1104786</Time_of_Day>
<Process_Name>ThreatProc.exe</Process_Name>
<PID>2728</PID>
```

```
<Operation>RegQueryValue</Operation>
<Path>HKLMS\Microsoft\Windows NT\...\
SurrogateFallback\Plane2</Path>
<Result>SUCCESS</Result>
<Detail>Type: REG_SZ, Length: 24, Data:
SimSun-ExtB</Detail>
</event>
```

Process Monitor allows to report system events for further analysis and reasoning to the PRONTOlogy module. Detailed report on system activities includes, but is not limited to:

- **process name** – the name of the process performing the operation;
- **operation** – the name of the operation being logged;
- **path** (if applicable) – the path of the object that the operation is performed on (e.g. a registry path, a file system path);
- **result** – the result of the operation (e.g. Success, EoF, Buffer Overflow);
- **detail** – additional operation-specific information about the event.

For the purpose of data acquisition it is also possible to use API hooking tools [32], [33], however, they inject themselves (like viruses) to the processes, thus they can affect results of the verification. In case of utilization of PRONTO malware hunting tool for detection of network attacks various network utilities, e.g. SNORT [34], ARAKIS [35], iptables [36], should also be used [37], [38].

Having stored CP-net models of cyber attacks in the database, it is possible to go further with the experiment to malware detection phase. As mentioned above, the aim of the experiment is not only to identify existing malware that was obfuscated, but also 0-day attacks that have, to some degree, similar behavior to the already identified one.

B. Malware detection scenario

Within one minute operation of Windows 7 OS thousands or even hundreds of thousands single events may be observed. Report from the Process Monitor includes everything that took place in the system. It includes both regular and suspicious activities.

For the purpose of verification and, in particular, generation of these *unwanted* activities, three different machines were infected by Virut, VBMania@MM, and 0-day attack that was simulated with events typical to different parts of malicious codes.

At the same time, various programs were executed on these three machines in order to simulate legitimate user activity. This allowed us to generate background regular events.

In the article we show only the first example and provide the reader with information on steps of PRONTO operation in terms of malware detection with emphasis on events filtering phase.

Let us assume that data acquisition phase allowed 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 following listing:

```
<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>HKLMS\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>
```

The events presented in above listing are processed and XML data is lifted to the semantic metadata. 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 rules that are valid in the presented scenario allow to infer that three of the above events are suspicious. These are the following rules:

```
Event(?e)^Place(?c) hasResource(?e,?y)^
resourceName(?y, "winlogon.exe")^
read(?y, ?z)^ ResFile(?z) ^
resourceName(?z,  "vrt7.tmp")->
hasPlace (?e,?c)^File(?c)^hasColour(?c, ?z)
```

```
Event(?e)^Place(?c)^hasResource(?e,?y)
^modify(?y, ?z)^ResRegistry(?z)^
resourceName(?z,"HKLMS\Microsoft\...\Run\
Windows System Monitor: C:\Windows\
system\winrsc.exe")->hasPlace(?e,?c)
^ Registry(?c)^hasColour(?c, ?z)
```

```
Event(?e)^Place(?c)^hasResource(?e,?y)
^connect(?y, ?z)^ResDomain(?z)
^resourceName(?z,  "irc.zief.pl")
-> hasPlace (?e,?c)^Domain(?c)^
hasColour(?c, ?z)
```

On the basis of these rules the following facts are inferred:

```
http://wil.waw.pl/secor/PRONTOlogy.owl#
Place_1 - 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 - 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 - 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 have been identified as suspicious, whereas event 3 – as a regular system activity. The SQWRL query that allowed to select this knowledge from the ontology had the following structure:

```
tbox:isSubClassOf(?subClass, Place)^
abox:hasIndividual(?subClass, x)->
sqwrl:select(?subClass)
```

```
Place(?p)^hasColour(?p, ?c)^
resourceName(?c, ?n)->sqwrl:select(?n)
```

The rules applied in the PRONTOlogy module allowed to pass 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 hasColour *object property*). It takes place in the *acquisition module* as presented in the architecture of solution. Then, in the PRONTOnet, these tokens are passed to *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`},
- $M_{Registry}$ = {`HKLMS\Microsoft\...\Run\Windows System Monitor:"C:\Windows\system\winrsc.exe"`}.

At the machine described in this scenario the detection realized with the use of CPN MM and marking M_a allowed to identify Virut attack. The result vector is as follows:

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

VII. Summary

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

Ontology presented for malware activities identification together with rules allows to filter out suspicious system activities and strongly supports malware detection mechanism implemented in PRONTO. The effectivenes of signature-based antivirus software is rapidly decreasing. Behavior based methods give promissing effects and should be investigated further on in modern security controls such as one presented here.

References

[1] A. Gostev, *Kaspersky Security Bulletin: Statistics 2008*, http://www.securelist.com/en/analysis/204792052/

[2] D. Maslennikov and Y. Namestnikov, *Kaspersky Security Bulletin. The overall statistics for 2012*, http://www.securelist.com/en/analysis/204792255/

[3] M. Conti, R. Di Pietro, L. Mancini, and A. Mei, "Mobility and cooperation to thwart node capture attacks in MANETs," *EURASIP J. Wirel. Commun. Netw.*, vol. 2009, no. 1, pp. 8:1–8:13, 2009. http://dx.doi.org/10.1155/2009/945943

[4] C. Raiu, *Virus News: 2012 by the numbers*, http://www.kaspersky.com/

[5] H. Tibbs, S. Ambler-Edwards, and M. Corcoran, *The Global Cyber Game: Achieving strategic resilience in the global knowledge society*, 2013, Defence Academy of The United Kingdom.

[6] S. Adair, R. Deibert, R. Rohozinski, N. Villeneuve, and G. Walton, *Shadows in the cloud: Investigating Cyber Espionage 2.0*, 2010, Information Warfare Monitor Shadowserver Foundation, http://shadows-in-the-cloud.net

[7] M. Szpyrka, B. Jasiul, K. Wrona, and F. Dziedzic, "Telecommunications networks risk assessment with Bayesian networks," in *Computer Information Systems and Industrial Management*, LNCS, Springer, 2013, vol. 8104, pp. 277–288. http://dx.doi.org/10.1007/978-3-642-40925-7_26

[8] P. Bereziński, M. Szpyrka, B. Jasiul, and M. Mazur, "Network anomaly detection using parameterized entropy," in *CISIM 2014*, ser. LNCS, K. Saeed and V. Snášel, Eds. Springer, 2014, vol. 8838, pp. 473–486.

[9] Z. Tarapata, M. Chmielewski, and R. Kasprzyk, "An algorithmic approach to social knowledge processing and reasoning based on graph representation – a case study," in *Intelligent Information and Database Systems*, LNCS, Springer, 2010, vol. 5991, pp. 93-104. http://dx.doi.org/10.1007/978-3-642-12101-2_11

[10] P. Szwed and P. Skrzyński, "A new lightweight method for security risk assessment based on fuzzy cognitive maps," *Applied Mathematics and Computer Science*, vol. 24, no. 1, pp. 213–225, 2014. http://dx.doi.org/10.2478/amcs-2014-0016

[11] *Verizone. 2012 Data Breach Investigations Report*, http://www.verizonenterprise.com/DBIR/2012/.

[12] A. Takeshi, K. Masaki, and T. Murakami, *Cyber Security Trend – Annual Review 2012*, http://www.nri-secure.co.jp/news/2012/pdf/cyber _security _trend_report_en.pdf

[13] *McAfee and HB Garry Solution Brief. Extend McAfee Total Protection for Endpoint with HBGary Digital DNA and Responder*, http://www.mcafee.com/us/resources/solution-briefs/sb- hbgary.pdf

[14] S. Bobek, K. Porzycki, and G. Nalepa, "Learning sensors usage patterns in mobile context-aware systems," in *Proceedings of the Federated Conference on Computer Science and Information Systems – FedCSIS*, IEEE, 2013, pp. 993–998.

[15] M. Szpyrka, "Exclusion rule-based systems – case study," in *Computer Science and Information Technology, IMCSIT*, 2008, pp. 237–242. http://dx.doi.org/10.1109/IMCSIT.2008.4747245

[16] M. Szpyrka, "Analysis of VME-Bus communication protocol – RTCP-net approach," *Real-Time Systems*, vol. 35, no. 1, pp. 91– 108, 2007. http://dx.doi.org/10.1007/s11241-006-9003-0

[17] G. Nalepa and S. Bobek, "Rule-based solution for context-aware reasoning on mobile devices," *Computer Science and Information Systems*, vol. 11, no. 1, pp. 171–193, 2014.

[18] K. Jensen and L. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, 1st ed. Springer, 2009.

[19] M. Szpyrka and T. Szmuc, "Decision tables in Petri net models," in *Rough Sets and Intelligent Systems Paradigms*, LNCS, Springer, 2007, vol. 4585, pp. 648–657. http://dx.doi.org/10.1007/978-3-540-73451-2_68

[20] B. Jasiul, M. Szpyrka, and J. Śliwa, "Malware behavior modeling with Colored Petri nets," in *CISIM 2014*, ser. LNCS, K. Saeed and V. Snášel, Eds. Springer, 2014, vol. 8838, pp. 667–679.

[21] G. Antoniou and F. van Harmelen, *A Semantic Web P rimer*. Cambridge, England: The MIT Press, 2008.

[22] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, *SWRL: A Semantic Web Rule Language. Combining OWL and RuleML*, http://www.w3.org/Submission/SWRL/

[23] J. Śliwa and B. Jasiul, "Efficiency of dynamic content adaptation based on semantic description of web service call context," in *Proceedings - IEEE Military Communications Conference MILCOM 2012, Orlando, USA*, 2012, pp. 1–6. http://dx.doi.org/10.1109/MILCOM.2012.6415810

[24] J. Śliwa, K. Gleba, W. Chmiel, P. Szwed, and A. Głowacz, " IOEM – Ontology engineering methodology for large systems," in *Computational Collective Intelligence. Technologies and Applications*, LNCS, Springer, 2011, vol. 6922, pp. 602–611. http://dx.doi.org/10.1007/978-3-642-23935-9_59

[25] A. Tarski, *Introduction to Logic and to the Methodology of Deductive Sciences, Second Edition*. New York: Dover Publications, Inc., 1946.

[26] J. Śliwa and M. Amanowicz, "A mediation service for web services provision in tactical disadvatnaged environment," in *IEEE Military Communications Conference, MILCOM*, 2008, pp. 1–7. http://dx.doi.org/10.1109/MILCOM.2008.4753323

[27] *Protégé – ontology editor and knowledge-base framework*, http://protege.stanford.edu/

[28] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," in *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, 2007, pp. 51 – 53. http://dx.doi.org/10.1016/j.websem.2007.03.004

[29] M. Russinovich and A. Margosis, *Windows Sysinternals Administrator's Reference*. Redmond, Washington, USA: Microsoft Press, 2011.

[30] *Microsoft Technet – Sysinternals*, http://technet.microsoft.com/en-us/sysinternals/

[31] M. Russinovich and B. Cogswell, *Process Monitor v3.05*, http://technet.microsoft.com/pl-pl/sysinternals/bb896645.aspx

[32] *API hooking revealed*, http://www.codeproject.com/Articles/2082/API-hooking-revealed

[33] *EasyHook*, http://easyhook.codeplex.com/

[34] *SNORT*, http://www.snort.org/

[35] *ARAKIS*, http://www.arakis.pl

[36] *Netfilter*, http://www.netfilter.org/

[37] B. Jasiul, R. Piotrowski, P. Bereziński, M. Choraś, R. Kozik, and J. Brzostek, "Federated Cyber Defence System – applied methods and techniques," in *2012 Military Communications and Information Systems Conference, MCC 2012*, 2012, pp. 145–150.

[38] M. Choraś, R. Kozik, R. Piotrowski, J. Brzostek, and W. Hołubowicz, "Network events correlation for federated networks protection system," in *Towards a Service-Based Internet*, LNCS, Springer, 2011, vol. 6994, pp. 100–111. http://dx.doi.org/10.1007/978-3-642-24755-2_9