An Ontology of Suspicious Software Behavior

March 22, 2016

[A]André Grégio Corresponding author: André Grégio, CTI Renato Archer, Rod. D. Pedro I (SP-65), KM 143.6, Amarais, 13069-901 - Campinas, Sao Paulo, Brazil., [A,B]Rodrigo Bonacin, [B]Antonio Carlos de Marchi, [A]Olga Fernanda Nabuco, [C]Paulo Lício de Geus Grégio et al. [A]Center for Information Technology Renato Archer (CTI), Campinas, São Paulo, Brazil
E-mail: {andre.gregio, rodrigo.bonacin, olga.nabuco}@cti.gov.br [B]FACCAMP, Campo Limpo Paulista/SP, Brazil
E-mail: carlosdmarchi@gmail.com [C]University of Campinas (Unicamp), Campinas, São Paulo, Brazil
E-mail: paulo@lasca.ic.unicamp.br

Abstract

–Malicious programs have been the main actors in complex, sophisticated attacks against nations, governments, diplomatic agencies, private institutions and people. Knowledge about malicious program behavior forms the basis for constructing more secure information systems. In this article, we introduce MBO, a Malicious Behavior Ontology that represents complex behaviors of suspicious executions, and through inference rules calculates their associated threat level for analytical proposals. We evaluate MBO using over two thousand unique known malware and 385 unique known benign software. Results highlight the representativeness of the MBO for expressing typical malicious activities.

Security ontologyMalware behaviorThreat analysis

1 Introduction

In today's Internet ecosystem—populated with a broad range of devices and systems, including physical plants—information security is increasingly required. Currently, Internet embraces a variety of complex and interconnected information systems including desktops, workstations, servers, smartphones, home appliances and, more alarming, platforms able to control critical infrastructure. The plethora of Internet-connected devices turn users into easy targets of computer device and network-related attacks. The main threat faced by information systems and their users today is malicious software (malware). Malware samples became very complex pieces of code that leverage a broad range of techniques to attack computer systems. These attacks aim to compromise systems in a way that the malware can install itself, establish remote access by its controller, bypass the security mechanisms, and finally, accomplish its objective. Cyber criminals have many motivations to write malware, e.g., stealing credentials, information, and other sensitive data, making profit, compromising third party systems etc. Since a malicious software is actually a program that will execute on a victim's system just like any other program, the intentions of a cyber criminal—from capturing credentials through functions that intercept the keyboard to political motivation—are translated into instructions, which will then be used to accomplish the infection. Each instruction can be seen as an action performed over a specific resource of the infected system and the resulting set of actions corresponds to the "behavior" of a malicious program. Knowledge about malware behavior is key to planning more secure systems and preventing future attacks.

Obtaining and analyzing behavior associated to malware is one effective way to understand infection procedures, measure the potential damage extent and assess defensive or protective measures. Therefore, malware behavioral analysis not only allows us to discover suspicious actions performed during an attack, but also to develop metrics to identify whether a program is behaving maliciously or not. However, the process of obtaining models that represent malware behavior is very delicate: it relies on constant monitoring of the victims resources, requiring a method to intercept all interactions among the process launched by a malicious program (and its child-processes) and subsystems of the target operating system. Furthermore, there is a need to screen out unsuspicious actions so as to focus on exhibited suspicious behavior. To do so, we need to previously know which actions are relevant to characterize a suspicious behavior, and how to decide if a given execution, among the many associated to processes running on a target system, is suspicious enough to raise an alert to a potential victim.

A path that may lead to a solution consists of a well defined scope of suspicious behavior, a core model and platform to share knowledge, metrics to calculate how suspicious the execution of a program is, and a specialized system that applies all these resources. By using such a system, users and specialists may interact—either by creating new rules or by refining existing ones—to make a collaborative system whose aim is to identify suspicious behaviors in potentially infected machines in an effective way, providing efficient update of the knowledge database. This way, it would be possible to address unknown malicious programs and maintain systems secure, even in cases where traditional security mechanisms (such as antivirus) are not able to detect threats.

Although preliminary malware ontologies do exist in the literature, they do not properly deal with current malicious programs. This happens due to i) malicious programs being traditionally classified into well-known, but too restrictive, categories (virus, worm, Trojan, bot), according to a predominant behavior—infects a file, spreads autonomously, disguises itself as a legitimate program and so on—and ii) existing malware ontologies leading to a hierarchy of classes (or categories, such as the ones mentioned above) that are far from adequate to address the issue of modern malware. This is especially true given that they are multipurpose, complex and may be split among several components. Moreover, as stated by [13], those class-based malware ontologies may "not be useful for malware instances that exhibit either behaviors from multiple classes or novel behaviors not associated with any recognized class".

Hence, there is a lack of a specialized ontology based on expert, collaborative knowledge that can handle suspicious behavior and, consequently, malware. In addition, there is the need for automatic inference of suspicious executions in monitored target systems. In this article, we introduce a novel malware ontology based on a set of commonly suspicious behaviors observed during years of malicious program analysis, as well as propose rules for inferring suspicious executions and to define the risk associated to a program for analytical purposes. Our main contributions are three-fold:

1. We present the conception of the core components of a Malware Behavior Ontology (MBO), which models our knowledge of malware behavior and establishes the foundations for further reasoning procedures.

2. We define rules with risk values for each suspicious activity contained in the hierarchy of main behaviors exhibited by malicious programs (introduced by [5]). The goal is to analyze program execution and attempt to characterize unknown malware processes based on the computed execution threat level.

3. We evaluate the proposed ontology using over 2 thousand malicious samples and almost 400 benign programs, testing the rules for suspicious executions and computing the associated risk. We also discuss the obtained results in terms of representativeness of the MBO, limitations of the approach and future directions.

The remainder of this article is organized as follows. In Section 2, we present related work on security-based and/or malware-inspired ontologies, as well as how they related to the work proposed in this article. In Section 3, we introduce the concepts behind the Malware Behavior Ontology (MBO), define the risks associated with each suspicious behavior, and present the inference rules created to evaluate monitored executions. In Section 4, we show a practical implementation of the proposed architecture and prototypes of modules that make use of MBO. We also show the results of applying MBO over a dataset of 2,245 malicious programs and 385 benign ones. In Section 5, we discuss the MBO application's obtained results, MBO limitations, future work and ongoing developments of this article's proposal. Finally, we conclude the article in Section 6.

2 Related Work

The difficulties related to malware analysis lie on discovering whether a chain of actions leads to a successful infection or not. Traditionally, the security community assigns "labels" to distinct types of malware according to their expected behavior. However, malware evolved significantly over the last 10–15 years, making those labels practically useless: current malware exhibit several of the classical behaviors at the same time. Moreover, antivirus vendors are migrating to heuristics that detect "generic" behavior, thus not assigning specific labels. Hence, an ontology for malicious programs is a demand to be supplied. While there are related ontologies, none of them addresses malware samples that exhibit multiple "suspicious" behaviors, which correspond to most of the current known threats. Below, we present ontologies closely related to malware found in the literature.

[6], for example, propose TWMAN (Taiwan Malware Analysis Net), a platform that aims to provide a knowledge and rule base, an ontology language, and analysis reports about malware samples. The TWMAN platform is composed of three layers—knowledge, communication, and application—to accomplish their proposal. In order to analyze malware for reporting, TWMAN uses a dynamic analysis component based on Truman Sandnet *truman. This component runs each sample and monitors some activities that may provide evidences of an infection, such as changes on special Windows Registry keys (Run and Service), network traffic generation, and modifications in the file system. IP addresses related to the malware execution, application-layer protocols and other associated data are gathered from network traffic analysis. The AIDE tool *AIDE is used to identify new files added to the victim's file system, as well as the modified or deleted ones. TWMAN's ontology structure is composed of a main object named "Thing" with four elements—"Malware_Impact_Target", "Malware_Type", "Malware_Behavioral", and "Malware_Sample". While "Malware_Impact_Target" and "Malware_Type" are limited to "Network", "Registry", or "File", and "Trojan", "Worm", or "Backdoor", respectively, the other elements are not described. However, TWMAN is limited by addressing only three malware types, which are not enough to describe the plethora of malicious behaviors. Furthermore, the ontology is not discussed in details in their work, which also affects the example choice (very simple) and the ontological structure (lack of precision).

Later, [7] proposed TWMAN+, which is based on Interval Type-2 Fuzzy Set instead of the Type-1 Fuzzy Set used in the previous version of the platform. Although the authors extended the model to embrace other malware types ("Botnet", "Viruses", "Rootkits"), the ontology domain is still limited, since their article does not present tests to validate the yield rates (and show how malware with multiple exhibited behaviors would fit within the fuzzy model). Also, some malware samples apply anti-analysis techniques in order to prevent dynamic analysis systems from monitoring their exhibited behaviors. [12] address malicious programs that make use of dormant behaviors, i.e., hidden malware behavior triggered only on special occasions (e.g., remote command, presence of specific applications installed in the target system, detection of analysis environment or security mechanism etc.). They propose to build these semantic models based on the observation of dynamic analysis traces and inference about the exhibition or not of a dormant functionality. However, the authors neither provide details about an actual semantic model, nor present any tests, results and their discussion.

Ontologies have also been used for detecting malware and Spam (unsolicited e-mail). [10] introduce an architectural model based on an intrusion ontology for this proposal (uCLAVS). Their approach consists of submitting files to multiple antivirus engines through a Web interface and, based on the returned outputs, detecting the file as malicious or not. These antivirus results are then used to create prevention rules that are the core of the proposed ontology, together with other results from attacks created by testing tools. Therefore, this ontology may be seen as a set of signatures to detect specific network attacks and malware samples that are already known by antivirus engines. While the authors define some inference rules based on the ontology's attributes (e.g., "Malware-Behavior", which serves as a flag to indicate whether a file is detected through submission to the antivirus engines or not), they do not give details about their ontology. A limitation in the employed detection mechanism is that the authors employ a committee of six standalone engines to decide whether a file is malicious or not, whereas they could use a publicly available service such as VirusTotal (`http://www.virustotal.com`), which allows remote users to test submitted files against $\approx$70 antivirus engines.

[15] discuss the problem of correctly classifying Spam and Ham (legitimate messages) due to high false alarm rates. To deal with this, the authors propose USpam, a Spam detection system based on an ontology that models users' interests regarding e-mail messages. USpam is composed of several modules to decompose messages, extract features, build the ontological model, and define required actions based on whether a message is detected as Spam or not. The major classes of USpam are "UserType", "SpamType", "UserInterests", "MessageType", and "MessageClasses". The class "SpamType" is composed of three subclasses that represent messages as "Abusive", "Malware", or "OutOfContext"; the class "UserInterests" allows more control to the user: if a message is not Spam for her, then the system will not apply the detection rules. The authors used machine learning algorithms (J48 decision tree and Naive Bayes) to generate an e-mail classifier, which obtained a false alarm rate of $\approx$10-30%.

The notion of suspicious behavior was explored by the PhishTester tool *Shahriar20121258. While this notion is close to that used in our work, their focus is on Web sites. PhishTester uses behavioral heuristics to decide whether a certain Web site is suspicious of being a phishing[1] or a legitimate one. To do so, they define rules that test the behavior of a Web site, i.e., compare the observed responses with a knowledge base of phishing and legitimate behaviors. Their set of behaviors is represented as finite state machines, which can result in legitimate or phishing states (e.g., a response such as "the site repeats the submission of the same form N times" can lead to phishing). The authors evaluate their proposal using 33 unique phishing URLs and 19 supposedly benign Web sites, correctly detecting all analyzed Web sites. More complex representation models are needed to deal with malware analysis, though.

[16] present an ontology based on fuzzy logic to address some known types

[1] An attempt to acquire sensitive information by disguising as a known, trustworthy entity.

of malware and their features. The ontology's superclass is named "Malact", which may be classified as "artifact"—virus, worm, botnet, spyware, backdoor, Trojan horse, rootkit and exploit—or "non-artifact". The ontology contains the description of four characteristics and their respective values: malware objectives; behavioral and technical features (from operational perspective); malware architecture and target's placement, which can be centralized, distributed, local, or remote; malware communication and management. For each of the artifact types, the ontology's application associates the values of the four available characteristics, thus producing a scheme that defines them. The relation among the chosen artifact types is used as an example application of the proposed ontology. An advantage of this ontology is to take the behavioral aspects of malware samples in more consideration than other approaches. However, the authors define a limited, general set of behaviors that does not address complex, multi-behavior malware.

Ontologies and detection rules were also explored by [8] in a method to identify malware activities. To discover malware, the authors developed a tool named PRONTO, whose aim is to filter system events (registry, process, file, network) and correlate them using predefined malware activities modeled using Colored Petri networks. Although the authors use an approach to collect execution data that is close to ours and a model for their "event" just like our "execution" class, there are no other similarities. The main difference is that they executed three malware samples from distinct families and, based on the monitoring results, used these malware event logs as a knowledge base for further detection rules, whereas we define dozens of suspicious behaviors that are independent of families. PRONTO seems to be an ongoing effort, since the authors introduced their architecture and ontology without an extensive evaluation.

[13] present the seed of a broader ontology focused on the cyber security domain. Their goal is to integrate data from different security-related sources and to reuse existing ontologies of the field, such as those describing attacks, vulnerabilities and malware. This ontology is based on the diamond model of malicious activities, which considers each corner as a threat dimension—victim, infrastructure, capability, actor. They emphasize the limitation of ontologies based on traditional malware classes to handle current, complex malware, which we intend to address in our proposal. [11] are working on an ontology for malware analysis whose goal is to provide a more scientific basis for exchanging incident information, training staff, creating related courses etc. To accomplish that, the authors built a malware analysis dictionary and taxonomy, defining a vocabulary to be used in the ontology. Since the malware analysis ontology is a preliminary work, the authors gathered some terms and sources to develop their work using OWL and Protégé, but did not provide details about the proposed hierarchy and classes.

Another related approach is presented by [2], who propose an ontology for mobile malware behavioral analysis. In their ontology the malware behavior is classified according to the damage, type of threat, infection routes and spreading mechanisms. However, the article does not provide enough details about how to model the complex sequence of malware actions; only the most abstract concepts are presented. What is more, their proposed model is applied in an end-user checklist for prevention purposes. Recently, [17] explored how ontologies can be used for analyzing threats and developing defensive strategies for mobile security. The authors justified the use of ontologies in face of the unidentified behavior exhibited by new mobile viruses. A set of viruses was analyzed in terms of the following virus behavior: "Sent SMS", "Information leaks", "Authority override", "Circumvented permissions", "Started service", "Broadcast receiver", "Operation links", "Outbound traffic", "Inbound traffic", "Encrypted API", "File read" and "File write". However, the article does not present a detailed declarative model, which can be collaboratively explored by users and experts. Two cyber-attack examples were used to demonstrate the approach, which were analyzed in terms of effectiveness in determining proper countermeasures.

Other related approaches include MAEC (`http://maec.mitre.org`) and MAL *mal, respectively a language to share detailed information about malware samples in a structured way and a malware-related dictionary of terms.

3 MBO - Malware Behavior Ontology

MBO was conceived as an attempt to represent a possible set of suspicious behaviors exhibited by malicious software. Before building the ontology's classes and their relationships, we had to identify traces of malware among benign programs running on the potential victim's operating system. From these traces, we extracted the existing behaviors according to those defined by [4]. The process of identifying malware traces was applied upon scenarios—not only competency questions—describing an attack over the Microsoft Windows system. For instance, a possible attack scenario would be a process that downloads known malware from the Internet and writes it over a legitimate system file. The creation of these scenarios allows us to identify the main attributes of a representative knowledge base of actions, behaviors, target systems, source systems and the possible sequence of occurrence of this behavior.

Ontologies can turn this task into a loosely coupled procedure, keeping the suspicious instance in a suspended state and relating the behavior to several suspicious processes at the same time, even if it is only a partial malicious behavior. Once a possibly-infected process is identified due to its suspicious behavior, a security agent can keep it as an instance of a distinct process and continue the analysis, instead of discarding it because it does not really fit any previously described behavior. Therefore, it would be possible to identify a malware sample that is unknown to antivirus or other defensive mechanisms.

3.1 Ontology-based cyber security model

The Malware Behavior Ontology—MBO—is based on the conceptual aspects introduced by [5]. Figure 1 presents an overview of MBO: its core classes, their relationships and data properties. The main concepts and relationships are described below, with the class names represented in italic.

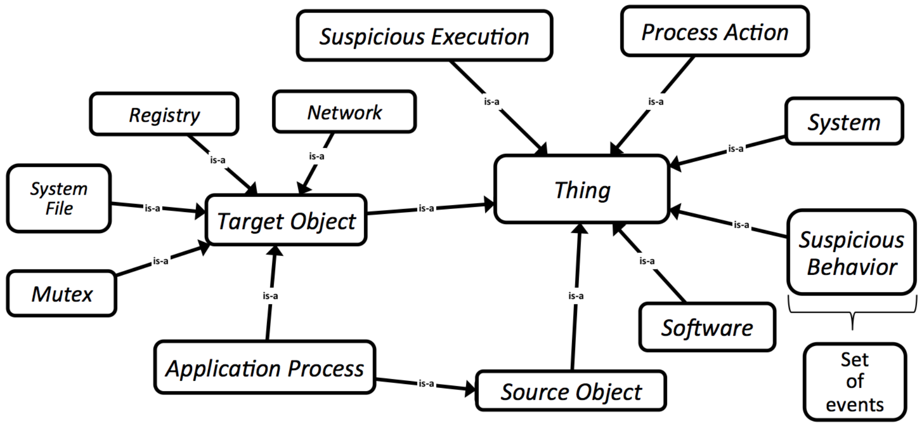


Figure 1: General view of the Malware Behavior Ontology (MBO).

SuspiciousExecution is the main class of the MBO. The relationships among classes are as follows:

- Each *SuspiciousExecution* should be associated with a *SuspiciousSoftware*;
- All *SuspiciousSoftware* instances are executed on a *System*;
- A *SuspiciousExecution* contains a set of *ProcessAction*s;
- A *ProcessAction* instance is associated with a timestamp, a process (with a unique name/identifier), a *SourceObject* (e.g., an *ApplicationProcess* that executes it), and a *TargetObject* (e.g., *Mutex*, *Network*, *Registry*, *SystemFile*, another *ApplicationProcess*);
- A *ProcessAction* instance may be linked to a *SuspiciousBehaviour* one, which can be one or more events (*AttackLaunchingEvent*, *EvasionEvent*, *RemoteControlEvent*, *SelfDefenceEvent*, *StealingEvent*, *SubversionEvent*)

Figure 2 illustrates the expanded *SuspiciousBehaviour* hierarchy of classes, containing all subclasses (events) and the behaviors pertaining to one or more of these events. These behaviors may then be specialized in suspicious activities, which provide more details about potentially dangerous actions performed during the infection life cycle.

To illustrate MBO, lets suppose a malware sample (its process was named after `mw.exe`) that performed the following activities observed during dynamic analysis, where bold-faced ones represent identified suspicious behaviors:

1. **mw.exe; create; mutex; xyz123**

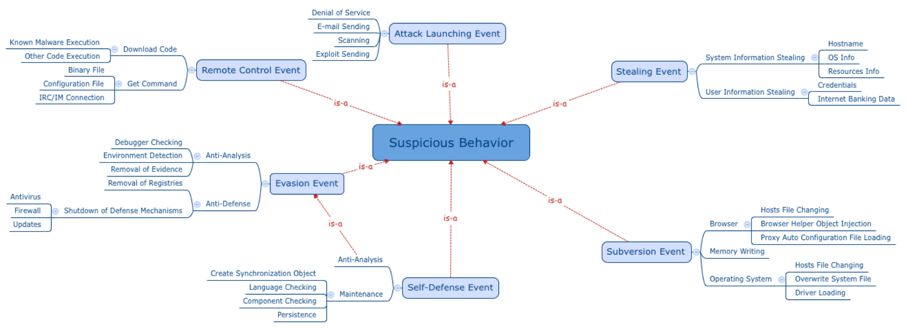


Figure 2: Expanded view of defined suspicious events, their associated behaviors and suspicious activities.

2. mw.exe; read; registry; ...\activecomputername\computername
3. **mw.exe; write; file; c:\windows\...\javaservice.exe**
4. **mw.exe; write; registry; ...\run\javaservice.exe**
5. **mw.exe; delete; file; c:\mw.exe**
6. mw.exe; terminate; process; c:\mw.exe

Mapping these logs to our ontology yields:

- `mw.exe` is an *ApplicationProcess* from the *SourceObject* class;
- `create`, `read`, `write`, `delete` and `terminate` are instances of *ProcessAction*;
- `mutex`, `process`, `file` and `registry` are instances of *TargetObject*;
- the action targets `xyz123`, `...\activecomputername\computername`, `c:\windows\...\javaservice.exe`, `...\run\javaservice.exe` and `c:\mw.exe` are values of *TargetObject*;
- Item 1 from the previous list exhibited a suspicious behavior belonging to the *Self-Defence Event* (*Maintenance*$\Rightarrow$*Create Synchronization Object*);
- Item 2 exhibited the first step of what may incur in a *Stealing Event*, since the malware process read the hostname, which is a system information. If this information is then sent through the network by `mw.exe`, then stealing materializes;
- Item 3 exhibited a suspicious behavior of *Remote Control Event* (*Download Code*$\Rightarrow$*Other Code Execution*);
- Item 4 exhibited a *Self-Defence Event* through a *Maintenance* behavior named *Persistence*;

- Item 5 exhibited an *Evasion Event* through an *Anti-Analysis* behavior named *Removal of Evidence*. This is also a *Self-Defence Event*. Notice that the target of item 5 is also an *ApplicationProcess*;

Figure 3 illustrates how the process action of Item 3 is instantiated by MBO: ellipses represent instances created due to the aforementioned process action; the names of the classes are identified above the ellipses; dashed lines identify the object properties; dashed boxes represent data property values. In the example from Item 3, a given suspicious execution has the process action *Beh1ac1* (unique identifier). As specified by the same Item, the action name is *write*. This process action has one source identified by *mw.exe* and has also a target object (a *SystemFile*) named *c:\windows\...\javaservice.exe*. This process action is also a *RemoteControlEvent*, since it exhibits this suspicious behavior.

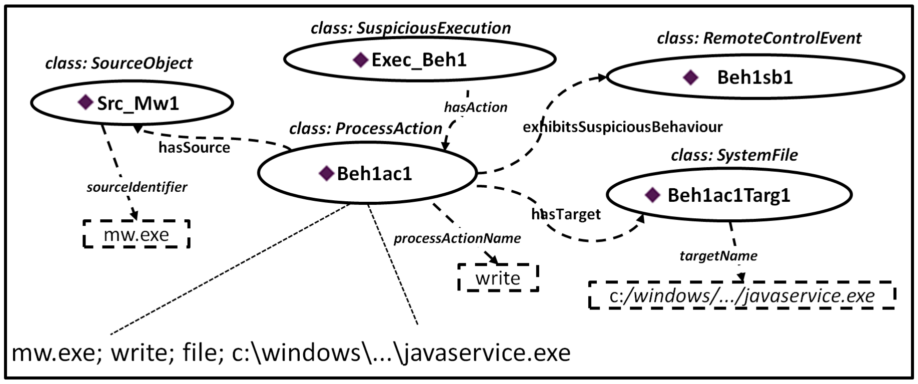


Figure 3: Example of a "ProcessAction" Instance.

A set of process actions with suspicious behaviors is the basis for modeling inference rules to determine whether an instance of *SuspiciousExecution* is linked to a malware sample or not. This work does not provide an off-the-shelf, ready-to-use solution, since it is an extensible model that defines rules and other parameters to automatically analyze potential malware with high level of precision. We expect to contribute with an ontological framework that is able to categorize and represent suspicious behavior in a precise way, which is a fundamental step for further developments on reasoning and detection procedures. In the next section, we present an initial set of rules for analyzing the risks associated to each execution.

3.2 Risks and Inference Rules

We based our inference rules and risk values on already defined behaviors extracted from malware captured in the wild. Most of these behaviors (and the associated rules derived from them) were described by [3], where the authors evaluated over 10 thousand malware samples to pinpoint malicious activities.

In order to evaluate how suspicious an execution is, we have to count the number of exhibited behaviors during each specific execution and how risky these behaviors are. To do so, we define a risk value for each of the behaviors ranging from 0 (absence of risky behavior) to 4 (critical behavior). The risk value assignment intends to produce a metric that helps us to identify potentially suspicious executions and, consequently, potentially malicious processes.

It is worth mentioning that it is out of this article's scope to produce a complete risk framework (e.g., NIST's Risk Management Framework[2]), and that the numeric values assigned to the behaviors are sufficient to our intent, since we consider the following risk values based on the activity's "impact" over the monitored system's security: 0 - no risk; 1 - low risk; 2 - medium risk; 3 - high risk; 4 - critical. This way, if an activity alone may be performed by a legitimate software or it is a behavior that usually occurs on "benign" executions, such as reading a Registry key containing information about the system, a low risk (value 1) is assigned to it. However, an activity that interferes with or deeply modifies the operating system kernel (e.g., loading a driver) or deliberately tampers with its security mechanisms (e.g., terminating the native firewall) is a critical one (risk value 4). The addition of browser add-ons/plugins, substitution of files and programs and removal of important registries are high risk behaviors (value 3), whereas other behaviors exhibited by benign and malicious software, such as those that involve communication with other systems, are considered of medium risk (value 2). The remaining possible behaviors, which are mostly ordinary actions not defined in our ontology, receive a risk value of 0, indicating that they are unsuspicious.

In Table 1, we present some of the behaviors defined in our ontology and their associated risks according to the above-mentioned rules.

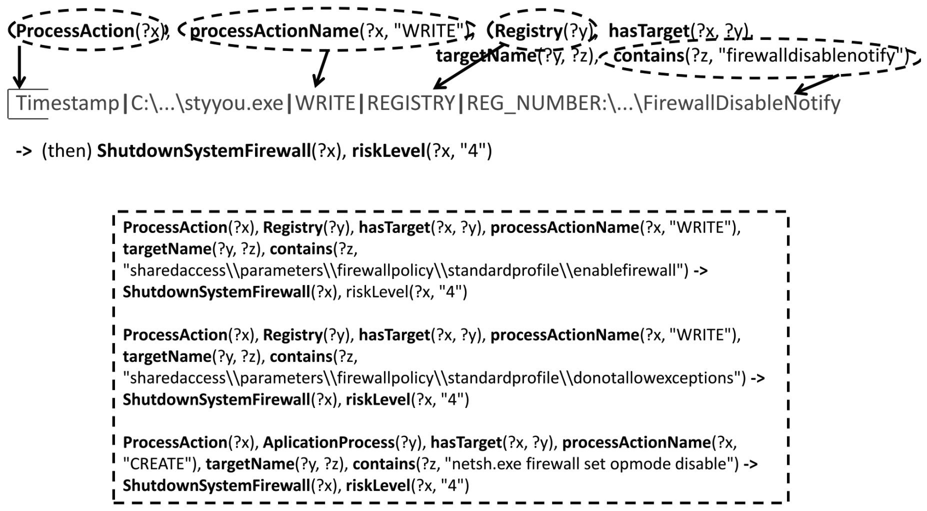
Figure 4 presents an example with four SWRL rules that identify the "ShutdownSystemFirewall" suspicious behavior. At the top, there is a rule and an example of process action from an execution log that exhibits this behavior. This figure highlights how each part of the rule is related to the log file as follows: the *ProcessAction(?x)* predicate is the entire line; *processActionName(?x, "WRITE")* is the third sentence element; *Registry(?y)* is the fourth one; and *hasTarget(?x, ?y), targetName(?y, ?z), contains(?z, "firewalldisablenotify")* is the fifth one. The rule head *ShutdownSystemFirewall(?x), riskLevel(?x, "4")* categorizes the process action as an instance of "ShutdownSystemFirewall" and attributes the *riskLevel* of 4 to this action (according to Table 1). *RiskLevel* is a DataProperty literal of a suspicious behavior. The subsequent rules are presented in the dashed box on the bottom. These rules express two typical actions that write on specific Registers and the creation of an application process to disable the firewall, respectively.

As a proof of concept, we created an initial set of 42 rules based on the subset of 18 suspicious behaviors with defined risks presented in Table 1, complying with the format presented in Figure 4. Our goal was to specify a rule set for identifying the most frequent behaviors, so as to enable initial experiments for

[2]http://csrc.nist.gov/groups/SMA/fisma/framework.html

Table 1: MBO subset of suspicious behaviors and their associated risks.

Suspicious Behavior	**Risk**
Evasion Event ⇒ Anti-Defense ⇒ Shutdown of Defense Mechanisms ⇒ Antivirus	4
Evasion Event ⇒ Anti-Defense ⇒ Shutdown of Defense Mechanisms ⇒ Firewall	4
Evasion Event ⇒ Anti-Defense ⇒ Shutdown of Defense Mechanisms ⇒ Updates	4
Subversion Event ⇒ Operating System and Browser ⇒ Hosts File Changing	4
Subversion Event ⇒ Browser ⇒ Proxy Auto Configuration File Loading	4
Subversion Event ⇒ Operating System ⇒ Driver Loading	4
Subversion Event ⇒ Operating System ⇒ Overwrite System File	3
Evasion Event ⇒ Anti-Defense ⇒ Removal of Registries	3
Subversion Event ⇒ Browser ⇒ Browser Helper Object Injection	3
Subversion Event ⇒ Memory Writing	2
Evasion and Self-Defense Event ⇒ Anti-Analysis ⇒ Removal of Evidence	2
Self-Defense Event ⇒ Maintenance ⇒ Persistence	2
Remote Control Event ⇒ Download Code	2
Remote Control Event ⇒ Get Command ⇒ [Drop] Binary File	2
Attack Launching Event ⇒ Exploit Sending	2
Remote Control Event ⇒ Get Command ⇒ IRC/IM Connection	2
Stealing Event ⇒ System Information Stealing ⇒ Hostname	1
Self-Defense Event ⇒ Maintenance ⇒ Language Checking	1
Self-Defense Event ⇒ Maintenance ⇒ Create Synchronization Object	1



```
ProcessAction(?x), processActionName(?x, "WRITE"), Registry(?y), hasTarget(?x, ?y),
                                      targetName(?y, ?z), contains(?z, "firewalldisablenotify")

Timestamp|C:\...\styyou.exe|WRITE|REGISTRY|REG_NUMBER:\...\FirewallDisableNotify

-> (then) ShutdownSystemFirewall(?x), riskLevel(?x, "4")
```

```
ProcessAction(?x), Registry(?y), hasTarget(?x, ?y), processActionName(?x, "WRITE"),
targetName(?y, ?z), contains(?z,
"sharedaccess\\parameters\\firewallpolicy\\standardprofile\\enablefirewall") ->
ShutdownSystemFirewall(?x), riskLevel(?x, "4")

ProcessAction(?x), Registry(?y), hasTarget(?x, ?y), processActionName(?x, "WRITE"),
targetName(?y, ?z), contains(?z,
"sharedaccess\\parameters\\firewallpolicy\\standardprofile\\donotallowexceptions") ->
ShutdownSystemFirewall(?x), riskLevel(?x, "4")

ProcessAction(?x), AplicationProcess(?y), hasTarget(?x, ?y), processActionName(?x,
"CREATE"), targetName(?y, ?z), contains(?z, "netsh.exe firewall set opmode disable") ->
ShutdownSystemFirewall(?x), riskLevel(?x, "4")
```

Figure 4: Rules to identify the "ShutdownSystemFirewall" suspicious behavior.

analyzing the consistence of the MBO regarding malicious and benign processes' log files (presented in the next section). This initial set can be expanded and refined in the future in a collaborative way to be broader and more precise.

4 Applying MBO

In this section, we describe the design and implementation of an architecture, a set of tools, and an experiment using an extensive set of malware execution logs, as well as a set of logs regarding non-malicious software to serve as a control set for the performed experiment. We also discuss some limitations of the deployed approach for analyzing threat levels to distinguish executions.

4.1 Architecture and Prototype

We defined a software architecture and set of tools to evaluate the use of the MBO and visualize the implications of using semantic technologies for malware analysis.

Figure 5 illustrates the overview of the implemented architecture, which consists of the following components (from bottom to top):

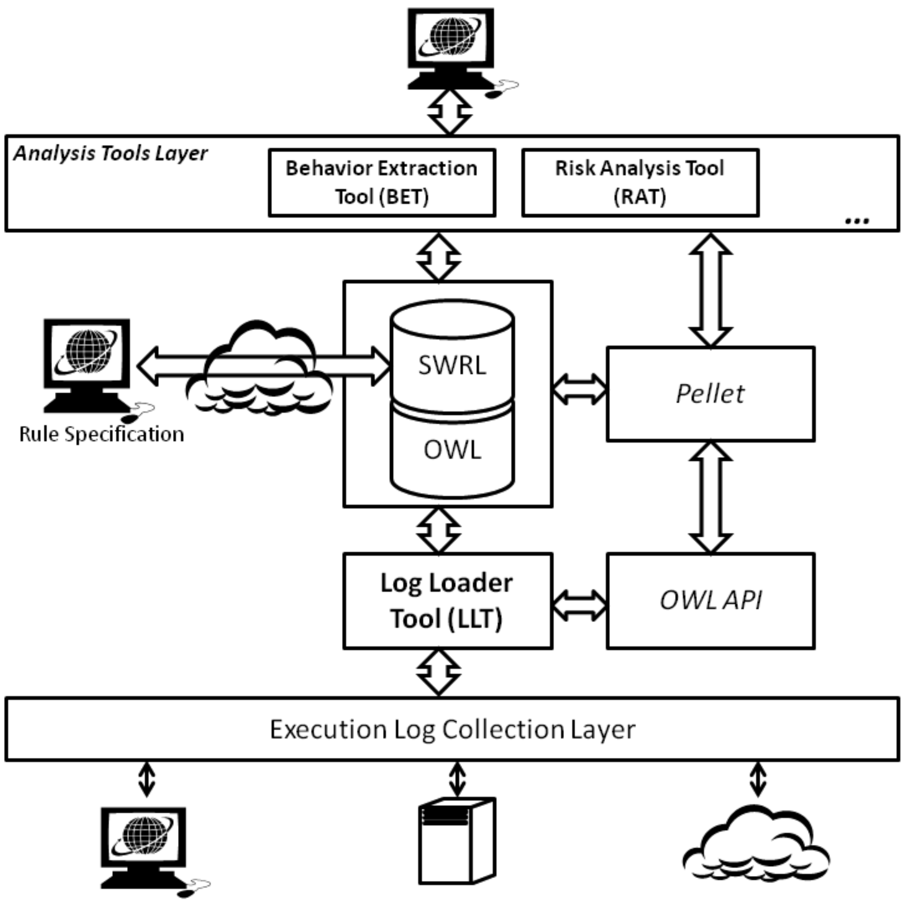


Figure 5: MBO Architectural Overview

- *Execution Log Collection Layer.* This component includes the tools, infrastructure and deployed procedures to collect data about malware and non-malware executions, which are presented in the next section.

- *Log Loader Tool (LLT).* This component reads the collected logs, performs consistency checking and includes all individuals and their respective properties into the ontology. The LLT uses the OWL API version 3 (for OWL 2.0) to manipulate the ontology. This tool works in a deterministic way, i.e., no complex classification algorithms were used: the LLT reads logs in a predefined format, indicating the content of each field in the log file as presented in Figure 3.

- *Rule Specification.* This component provides interfaces to specify SWRL. From a conceptual point of view, multiple interfaces can implement a collaborative infrastructure for crowd-sourcing rule specification. In the implemented experiment, the rules were specified directly on the Protégé interface.

- *Analysis Tools Layer.* This abstract component provides a set of tools that enables the inclusion of multiple tools for malware analysis using the "populated" ontology. For this experiment, we implemented two tools as follows:

  - *Behavior Extraction Tool (BET).* This tool focuses on the generation of statistics related to the modeled behavior occurrences. The user can setup the behavior to be analyzed on multiple hierarchical levels, according to the ontology structure. BET reads a set of execution logs that has already been instantiated on the ontology by LLT, the set of SWRL rules, the class axioms that determine the possible behaviors and performs inferences using the Pellet 2.2 API to determine the occurrence of each behavior from the set. This is a rule-based analysis performed by executing SWRL code that determines conditions for associating suspicious behavior to each action (see Figure 4). Section 4.6 presents results of this analysis.
  - *Risk Analysis Tool (RAT).* This tool focuses on the analysis of the set of logs and presents statistics about the observed execution behaviors, as well as the risk levels in an individual fashion. RAT reads a set of executions already instantiated on the ontology by LLT, the set of SWRL rules (e.g., those presented in Figure 4) and class axioms—including those that determine the execution behavior and risk levels—and performs inferences using the Pellet 2.2 API. It is possible, using RAT, to visualize statistics about risk levels associated to malware executions. Section 4.7 shows a use of RAT to analyze the potential of our ontology to represent threat levels of malware and non-malware executions.

4.2 Experiment Goals

This experiment was conducted with the following goals:

- to evaluate the representativeness of the modeled behaviors regarding the available dataset of malware samples;
- to analyze each malware exhibited behavior, including multiple incidences of suspicious behavior on each malware execution, and compare them with non-malware observed behavior;
- to present evidences about the potential of the proposed ontology to describe threat levels of program executions;
- to investigate practical implementation issues on an Internet-distributed solution.

4.3 Experiment procedure

Our experiment begins with the collection of representative samples of malicious software and the observation of how they interact with the victim's operating system and network, in order to obtain their execution traces. To do so, we collected 2,245 malware samples from the following sources: honeypots[3], phishing e-mails[4] and research collaborations[5]. Since there is no ground truth (gold standard) to represent the diversity of all existing malware samples, our collection may be considered representative enough of current (as of 2014/2015) malicious software seen in the wild, therefore attacking real victims. Regarding non-malicious software, we used 385 executable files inherently present on a pristine version of the Windows operating system, since they are supposed to be benign despite some of them being able to perform privileged and/or administrative operations on the system. Literature on malware classification is broad, but most of them consider a dataset of only malicious samples *bailey2007,Rieck2008,bayer09,Park2010,dimva2012, which may raise concern about the quality of the results. Other problems in reproducing results from related works include the use of private datasets and innate classification problems, such as biased datasets, easy-to-detect malware samples, bad frequency distribution of malware families etc. *Li:2010:CEM:1894166.1894183. [14] evaluate the importance of following rigorous procedures to prevent these issues.

After collected, we submitted these samples to a dynamic analysis system to have their execution traces extracted. We ran our obtained malware samples at the same time we monitored the target operating system and network activities using the techniques proposed by [1], which resulted in logs containing the

[3]Security sensors whose purpose is to be probed, attacked and compromised so that they are able to collect logs, malicious code and attack data to provide useful information for researchers.

[4]E-mail with links to infected Web pages that download malware when accessed or that contain malicious attachments.

[5]Malware samples donated by other researchers or institutions.

activities performed by the malware under analysis during the infection step. This dynamic analysis system consists of a QEMU's *qemu emulated operating system (Windows XP SP3) with a kernel driver that hooks native system calls, monitoring the actions launched by the analyzed malicious process and all its spawned child processes. Hence, if a malicious process tries to subvert a system process, this action will be logged, as well as every single action the newly-compromised process performs. From outside of the emulated environment, we capture all the network traffic produced by the malicious execution, thus providing additional information that allows us to identify potentially suspicious behaviors.

Finally, the analysis tools were executed with multiple parameters to provide the experiment results. The executions were performed in 10 blocks of samples due to memory consumption. The final results were summarized after the execution. After the traces have been extracted due to the execution of samples in the dynamic analysis system, we launch BET to identify the occurrence of each behavior on malware samples. We then execute RAT to calculate the threat level associated with the monitored execution.

4.4 Environment and Limitations

All of the aforementioned tools were built using the JEE 8 platform and executed under Windows 7 installed on a desktop computer having an Intel Core i7 processor and 16 GB of RAM. After the rule specification, the LLT was executed, thus producing a populated ontology. The implementation of the architecture used in this experiment was limited to detect the 18 suspicious behavior classes presented in Table 2. Some classes were represented by their superclasses, since additional studies are required to determine a complete rule detection set. In this sense, the results presented in the next sections are limited by the implementation of the architecture.

4.5 Analysis Procedure

The first step of the analysis was based on BET results and was aimed at testing the following hypothesis: *the modeled behaviors are typical malware execution actions.* Consequently, we focused on the occurrence of each behavior on malware samples. This is especially important when we need to determine if the behavior categories modeled in the ontology are actually exhibited during malware executions. The occurrence of the defined behaviors on malware executions was then compared with their occurrence on non-malware execution, so that we can verify if the modeled behavior would be "typical" of malware executions (and not "typical" of non-malware executions). The relative frequency of each behavior also indicates the most frequent ones, and in some cases may indicate the need for modeling specialized classes.

The second step of the analysis was based on RAT results and was focused on the following hypothesis: *the modeled behaviors and rules are able to characterize the threat level of malware executions.* First, we analyzed the distribution of the executions according to the calculated threat levels. As previously presented (Table 1 in Section 3.2), a weighted value was attributed to each rule to represent its associated risk level. In this experiment, we summed the values of each rule triggered on execution to calculate an execution threat level t. We classified each execution based on four threat levels: "No Threat", for executions that had $t = 0$; "Low Threat", for executions where $1 \leq t < 5$; "Medium Threat", for executions where $5 \leq t < 20$; and "High Threat", for executions that have $t \geq 20$. We also calculated the "Mean Threat Level" for malware and non-malware executions and the standard deviation of the samples executions.

Although it is not our intention to deal with malware binary classification issues, additionally, we also calculated the precision (positive predictive value), sensitivity (recall), specificity, accuracy and Matthews correlation coefficient measure (MatthewsCC)[6] using the following equations:

$$Precision = \frac{TrueP}{TrueP+FalseP}$$

$$Sensitivity = \frac{TrueP}{TrueP+FalseN}$$

$$Specificity = \frac{TrueN}{TrueN+FalseP}$$

$$Accuracy = \frac{TrueP+TrueN}{TotalPopulation}$$

$$MatthewsCC = \frac{(TrueP*TrueN)-(FalseP*FalseN)}{\sqrt{(TrueP+FalseP)*(TrueP+FalseN)*(TrueN+FalseP)*(TrueN+FalseN)}}$$

where:

TrueP is the number of malware samples with t ≥ 1,

FalseP is the number of non-malware samples with t ≥ 1,

TrueN is the number of non-malware samples with t < 1,

FalseN is the number of malware samples with t < 1,

TotalPopulation is the sum of malware and non-malware samples.

4.6 Malware Behavior Occurrence Analysis

Table 2 intends to synthesize the results obtained using BET. Its columns presents (from left to right):

- a list of the tested suspicious behaviors;
- the absolute number of occurrences of each behavior, this behavior's percentage over the total detected behaviors (13,906) and the number of oc-

[6]Aims to consider malware and non-malware different samples' sizes.

currences divided by the number of analyzed executions (2,245) for all malware executions;

- the absolute number of occurrences of a behavior, the percentage of this behavior from the total of detected behaviors (429) and the number of occurrences divided by the number of analyzed executions (285) for all non-malware/benign software;
- finally, the last column shows the ratio between occurrences per malware execution and occurrences per non-malware execution.

Table 2: Behavior analysis using BET, ordered by ratio. This may already tell which behaviors would allow a better classification.

Suspicious Behavior	**Malware**			**Non-Malware**			**Ratio**
	# occur.	**% occur.**	**Occ./Exec.**	**# occur.**	**% occur.**	**Occ./Exec.**	
ShutdownSystemFirewall	73	0.52	0.03	0	0.00	0.00	∞
ShutdownUpdates	50	0.36	0.02	0	0.00	0.00	∞
ShutdownAntiVirus	48	0.35	0.02	0	0.00	0.00	∞
ProxyAutoConfigurationFileLoad	42	0.30	0.02	0	0.00	0.00	∞
LanguageChecking	10	0.07	0.00	0	0.00	0.00	∞
BrowserHelperObjectInjection	5	0.04	0.00	0	0.00	0.00	∞
ExploitSending	3	0.02	0.00	0	0.00	0.00	∞
DriverLoading	2	0.01	0.00	0	0.00	0.00	∞
HostsFileChanging	2	0.01	0.00	0	0.00	0.00	∞
IRC/IMConnection	2	0.01	0.00	0	0.00	0.00	∞
Persistence	574	4.13	0.26	1	0.23	0.00	72.87
DownloadCode/DropBinaryFile	1768	12.71	0.79	15	3.50	0.05	14.96
MemoryWriting	1646	11.84	0.73	26	6.06	0.09	8.04
OverwriteSystemFile	1943	13.97	0.87	35	8.16	0.12	7.05
RemovalOfEvidence	2591	18.63	1.15	54	12.59	0.19	6.09
CreateSynchronizationObject	3231	23.23	1.44	179	41.72	0.63	2.29
HostNameStealing	1916	13.78	0.85	119	27.74	0.42	2.04

Figure 6 illustrates the frequency of suspicious behaviors detected by BET in the malware samples set. A small set of seven behaviors presented on the left side of Figure 6 corresponds to 98.29% of all the identified behaviors exhibited on malware samples executions, whereas the remainder behaviors correspond to 1.71%. These most frequent behaviors also correspond to 100% of the ones found on non-malware samples.

"CreateSynchronizationObject" is the most frequent behavior found, corresponding to 23.23% (3,231) of occurrences. This means 1.44 occurrence per execution (Table 2), considering all available malware executions. "CreateSynchronizationObject" is also the most frequent behavior found on non-malware executions, with 41.72% of the occurrences. However, this represents 0.63 occurrences per execution, which indicates that this type of behavior is 2.26 times more frequent on malware (rightmost column of Table 2). The "HostNameStealing" is the behavior with the smaller difference on both (malware and non-malware) occurrences, since it is 2.04 times more frequent on malware executions than on benign ones. The small difference occurs due to the fact that both behaviors

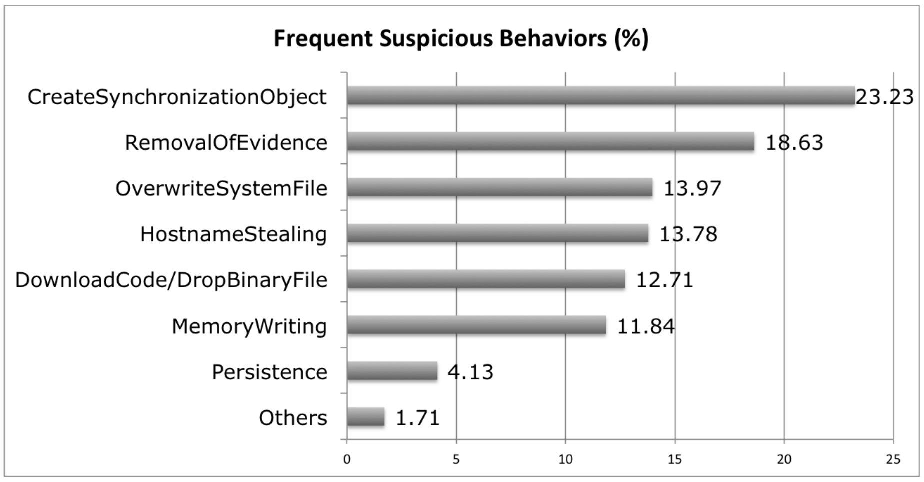


Figure 6: The most frequent suspicious behaviors detected on malware sample execution logs.

may appear frequently either in benign programs or in malicious ones: a synchronization object (a.k.a. mutex or semaphore) may serve as an indication that a program is already running, so another instance of the same program is not launched; before "stealing" the hostname data, i.e., sending it to a remote computer, an application needs to read it from the corresponding Registry, which triggers our pattern for this behavioral matching.

Other differences seen at the Table 2 rightmost column show that even when applying simple patterns, we are able to separate between malware and non-malware. Behaviors whose difference is small (in addition to the aforementioned "CreateSynchronizationObject" and "HostNameStealing") may be explained due to the reason behind their exhibition among non-malware programs. "RemovalOfEvidence" (6.09) happens when an application deletes itself or another program created during its execution. This behavior is also exhibited by legitimate installers, which creates some temporary files to extract the program the user intends to install and then deletes itself and the temporary files. "OverwriteSystemFile" (7.05) is a behavior common to the updating/patching process, since a library or the updated program may be substituted by a newer, fixed version. "MemoryWriting" behavior (8.04) is exhibited if a monitored program calls another program or process in order to execute it with some parameter (or in background). It is not unusual that legitimate programs need to call `cmd.exe` or a browser due to some required functionality. Installers may also explain the non-malware occurrence of "DownloadCode" and "DropBinaryFile" suspicious behaviors (difference = 14.96), but all program installations must be thoroughly observed by users or security mechanisms. "Persistence" is a behavior highly frequent on malware (for maintaining the infectious process running and/or a remote point of access to the attacker), but uncommon in non-malware (except for agents or instant messaging-like programs). In this case, there is a huge difference (72.87) between both types of software.

The obtained results suggest that we need to study these specific behaviors in detail in order to find out for which circumstances such behaviors occur on malware and non-malware. The study may lead to a more complex ontology and rule set that will allow us to extend the MBO, as opposed to contradict it. The evidences also suggest that there is a considerable number of behaviors that appear only on malware executions, however these behaviors are not enough to characterize a program as malicious due to the low frequency of occurrences. The remaining behaviors (for which the difference could not be calculated due to their absence in the non-malware analyzed) are not commonly observed in "benign" programs, unless in very specific cases. Just to name a few: the automated shutdown of protective mechanisms (antivirus, firewall, updates) should never be exhibited by non-malware[7]; loading a proxy auto configuration file may be performed by a network administrator in a company, but is a very unusual activity to be performed by an ordinary user or a standalone program (malware abuses this to redirect users' browsing to cloned sites); injecting a browser helper object is the equivalent of installing a browser's extension or plugin, which is also an uncommon behavior for a benign software. In general, the identified behaviors were 4.17 times more frequent on malware samples, with 6.19 occurrences per malware execution. These results indicate that an ontology-based approach has the potential to improve malware behavioral analysis.

4.7 Malware Threat Level Computation and Limitations

In this section, we show how we conducted an experiment using RAT. From the total of 2,245 malware executions, we considered 1,672 executions that presented successful attacks. The remaining 573 executions did not present any behavior other than to terminate their own process. On the one hand, this means the associated 573 malware samples may have realized that they were under analysis, thus either simply quitting or exhibiting a split, unsuspicious behavior *split. On the other hand, these samples could be easily detected as suspicious by applying a simple pattern matching approach to search for embedded anti-analysis techniques *chen2008towards*rubirabranco, influencing (positively) our results. Therefore, we removed these specific executions. The entire group of non-malware executions (385 samples) was considered for our tests.

Table 3 shows the distribution of Malware and Non-Malware executions according to the calculated threat level. In 95.51% of the cases, the analyzed malware execution presented some threat level. However, in 4.49% (75) of the cases, RAT was not able to identify any threat. On the one hand, this number could be reduced by an implementation that considered a broader range of behaviors (we specified rules for 18 suspicious behaviors in this experiment). On the other hand, RAT identified high threat levels for non-malware execution in

[7]Even if a legitimate software requires a temporary disabling in order to be installed, it is up to the user perform the task manually.

six cases (2.11%). More complex rules should be deployed to characterize the context of these exhibited suspicious behavior, minimizing the misleading high threat levels. However, it is entirely possible that benign software presents real suspicious behaviors, which could be evaluated by any user of MBO.

Table 3: Threat level calculation using RAT

Threat Level	**Malware**		**Non-Malware**	
	# exec.	**% exec.**	**# exec.**	**% exec.**
No Threat ($t = 0$)	75	4.49	177	62.11
Low Threat ($1 \leq t < 5$)	585	34.99	76	26.66
Medium Threat ($5 \leq t < 20$)	560	33.49	26	9.12
High Threat ($t \geq 20$)	452	27.03	6	2.11
Mean Threat Level	16.37		1.81	
Std. Deviation	31.75		4.58	
Precision	0.94			
Sensitivity	0.96			
Specificity	0.62			
Accuracy	0.91			
MatthewsCC	0.61 (-1 to 1)			

In addition, Table 3 presents the average threat level of Malware executions (16.37) and the standard deviation of the obtained results (31.75), as well as the average threat level of Non-Malware executions (1.81) and their related standard deviation results (4.58). It is worth noting that the threat level for Malware executions is 9 times higher than for Non-Malware executions. In general, RAT is able to significantly distinguish between the two types of execution (see Figure 4.7), but our results also point to limitations that need to be addressed in future work, as discussed in the next section. Finally, Table 3 also presents some typical binary classification measures. The results show good precision (0.94), sensitivity/recall (0.96) and accuracy (0.91) values, while the specificity is 0.62 and MatthewsCC value is 0.61. Binary classification is not the objective of this paper, and the application was not tuned to increase the classification performance. For example, it is possible to increase the MatthewsCC value to 0.73 (with a sensitivity of 0.94 and a specificity of 0.84) if we just remove the "HostNameStealing" behavior. In the context of this paper, "HostNameStealing" is a "StealingEvent" that can be important for malware behavior analysis and to identify the threat level of an execution (despite the penalty in the classification performance), since malicious software may use this information for several attacks (e.g., identity spoofing/impersonation and assets enumeration). However, legitimate software, such as operating system updating mechanisms, may also make use of this action. This may indicate that more specialized concepts and rules could be modeled in our ontology to increase the distinction between malware and non-malware executions.

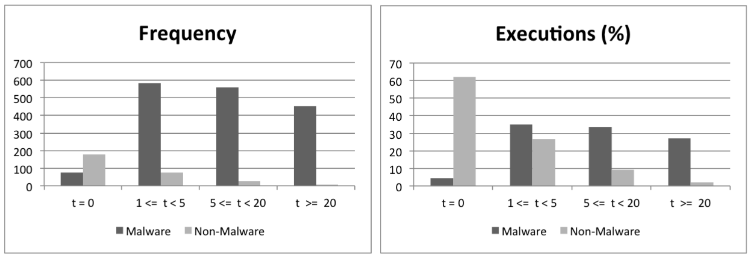


Figure 7: Comparison between the threat level of malware and non-malware based on frequency (left) and percentage (right) of executions using RAT.

5 Discussion about MBO

The experiments shown in the previous section suggest promising results on identifying programs behaving suspiciously through analyzing software executions. The analysis intends to find suspicious behaviors and calculate the execution's threat level based on the risk associated to each behavior defined on MBO. In this section, we discuss advantages, future works, challenges and limitations of the proposed Malware Behavior Ontology[8].

One of the main advantages of the MBO is the fact that it is extensible, i.e., we can expand the ontology to deal with additional or novel malware-related behaviors. Due to this extensibility, it is possible to be up-to-date with new malware features observed in the wild. In addition, we can deploy MBO as a collaborative system, which may also be modified to act as a distributed, real time evaluation tool on client machines. Since the ontology uses a URI-based structure, it can be deployed on the Web and, when there are any changes (e.g., new behavior inserted, inference rule modified etc.), all "agent tools" installed on clients will be automatically updated without the need for file downloading.

Furthermore, the rule specification procedure is very flexible: the rules that determine the behaviors, as well as the risk and threat levels, can be collaboratively built and maintained. It is also possible to build complex rules and inference chains so as to model more sophisticated suspicious behaviors. SWRL allows the construction of these complex rules and inference chains, which may be used to classify programs based on their malicious behavior or even to detect malware. A MBO user can learn if an initially "benign" software has been presenting suspicious behaviors, being able to analyze it to identify how malicious the evaluated software actually is.

The current limitations of our approach include performance issues: during the performed tests, the memory heap size increased 1GB when BET was running, and 800MB when RAT was running, for each set of around 80–100 (depending on the average size) execution logs under analysis. We solved this issue by dividing the 2,245 logs into small subsets and combining the calculated

[8]Sample OWL file available at `http://bit.ly/1VnzTyx`.

results at the end of the analysis procedures. Although this is a viable solution, we need a more sophisticated architecture to deploy it in the real world, such as grid-based solutions and database support *DBLP:journals/fgcs/SyedSB13. Although Ontologies and rules are extensible from a modeling perspective (i.e., new classes, axioms and rules can be created to extend the model), the scalability is limited by inference engines. The Semantic Web community has produced tools to analyze the performance of rule-based engines *Liang2009 and also strategies to optimize rules, which should be considered when ontologies have a large number of instances. The MBO's extension also hints on the adoption of safe rules[9] to prevent situations in which OWL DL and SWRL are undecidable. We leave these and other scalability and extension issues (distributing the engines to run/monitor programs and produce execution logs) as future work.

MBO, its architecture, and related tools were not built to detect malware in real time. Due to this fact, we observed some false positives that were not addressed in this article. Although our observed results were promising to use in malware detection systems, many aspects should be considered for its adoption as a real-time tool, such as how to deal with false-positives (handling non-malicious software regarding their suspicious behaviors) and false-negatives, how to detect those situations, and, consequently, how to establish the threat level. Steps toward a solution include i) the creation of white lists of common targets, resources and behaviors, ii) taking into account users' comments and collaborative feedback, and iii) relying on some sort of crowd-sourcing solution to rank executions, attributing weights to more frequent labels (suspicious, benign or malicious).

It is worth mentioning that we do not attempt to classify an execution as "virulent" or malicious. Instead, we provide a threat level and behavioral information that may be useful to help in a decision taking process. To improve our approach, we could enhance behavior rules by adding self-adaptiveness. Moreover, machine learning techniques could be applied to adapt the active ruleset when new threats arise. However, such a solution demands efforts on multiple fields and technologies.

Finally, MBO can be extended to include more specific security aspects, for instance, behaviors that point to execution sequences threatening users' privacy, among others. The possible extensions include how to model the relations of program's instructions to attacker's intentions, and how the intentions are actually translated into instructions. This demands further research on multidisciplinary fields including cognitive sciences, cyber security and ontology engineering. Studies of human cognition are also a promising strategy to improve human decision on secure environments *Endsley2012,Gonzalez2014, by presenting alternatives, for example, to represent the cognitive process, organize information and reduce the information workload.

[9]`https://km.aifb.kit.edu/ws/prowl2006/prowl06_4on1.pdf`

6 Conclusion

In this article, we proposed MBO, an ontology built upon the knowledge about suspicious behaviors commonly exhibited by malicious programs. We defined a hierarchy of classes comprising suspicious events and their associated behaviors, described each of them and attributed a risk level that serves as a weight to calculate the threat level of a monitored execution. We designed an architecture for the proposed ontology, developed tools to collect execution logs, parse those logs, extract behavioral information, apply the inference rules we build to identify suspicious executions and, finally, calculate the threat level of each analyzed execution. To validate our proposal, we tested MBO using 2,245 unique malware executions collected from actual samples and 385 benign executions from legitimate operating system programs. The obtained results showed that our approach is promising and can be potentially used in real-time malware detection on client hosts. As for future work, we intend to develop a collaborative framework to receive/refine rules and to update the ontology, benefiting from expert knowledge or crowd-sourcing feedback, and sharing malware detection related information.

References

[1] Afonso, V. M., Filho, D. S. F., Grégio, A. R. A., de Geus, P. L., and Jino, M. A hybrid framework to analyze web and os malware. In *Proceedings of the IEEE Internactional Conference of Communications (ICC)* (Ottawa, Canada, June 2012).

[2] Chiang, H.-S., and Tsaur, W.-J. Mobile malware behavioral analysis and preventive strategy using ontology. In *SocialCom* (2010), A. K. Elmagarmid and D. Agrawal, Eds., IEEE Computer Society, pp. 1080–1085.

[3] Grégio, A. R. A., Afonso, V. M., Filho, D. S. F., de Geus, P. L., Jino, M., and dos Santos, R. D. C. Pinpointing malicious activities through network and system-level malware execution behavior. In *12th International Conference on Computational Science and Its Applications (ICCSA). Lecture Notes in Computer Science* (Salvador, BA, Brazil, June 2012).

[4] Grégio, A. R. A., Afonso, V. M., Filho, D. S. F., de Geus, P. L. d., and Jino, M. Toward a taxonomy of malware behaviors. *The Computer Journal 58*, 10 (2015), 2758–2777.

[5] Grégio, A. R. A., Bonacin, R., Nabuco, O., Monte Afonso, V., Lício De Geus, P., and Jino, M. Ontology for malware behavior: A core model proposal. In *WETICE Conference (WETICE), 2014 IEEE 23rd International* (2014), IEEE, pp. 453–458.

[6] Huang, H.-D., Chuang, T.-Y., Tsai, Y.-L., and Lee, C.-S. Ontology-based intelligent system for malware behavioral analysis. In *FUZZ-IEEE* (2010), IEEE, pp. 1–6.

[7] Huang, H.-D., Lee, C.-S., Hagras, H., and Kao, H.-Y. Twman+: A type-2 fuzzy ontology model for malware behavior analysis. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on* (Oct 2012), pp. 2821–2826.

[8] Jasiul, B., Sliwa, J., Gleba, K., and Szpyrka, M. Identification of malware activities with rules. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014.* (2014), M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., pp. 101–110.

[9] Kott, A., Wang, C., and Erbacher, R. F., Eds. *Cyber Defense and Situational Awareness*, vol. 62 of *Advances in Information Security.* Springer, 2014.

[10] Martinez, C. A., Echeverri, G. I., and Sanz, A. G. C. Malware detection based on cloud computing integrating intrusion ontology representation. In *Communications (LATINCOM), 2010 IEEE Latin-American Conference on* (September 2010).

[11] Mundie, D. A., and McIntire, D. M. An ontology for malware analysis. In *2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg, Germany, September 2-6, 2013* (2013), IEEE Computer Society, pp. 556–558.

[12] Noor, M., and Abbas, H. Anticipating dormant functionality in malware: A semantics based approach. In *Biometrics and Security Technologies (ISBAST), 2013 International Symposium on* (July 2013), pp. 20–23.

[13] Obrst, L., Chase, P., and Markeloff, R. Developing an ontology of the cyber security domain. In *STIDS* (2012), P. C. G. da Costa and K. B. Laskey, Eds., vol. 966 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 49–56.

[14] Rossow, C., Dietrich, C. J., Kreibich, C., Grier, C., Paxson, V., Pohlmann, N., Bos, H., and van Steen, M. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook . In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)* (San Francisco, CA, May 2012).

[15] Shoaib, M., and Farooq, M. Uspam – a user centric ontology driven spam detection system. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on* (Jan 2015), pp. 3661–3669.

[16] Tafazzoli, T., and Sadjadi, S. H. Malware fuzzy ontology for semantic web. *International Journal of Computer Science and Network Security 8* (2008), 153–161.

[17] Wang, P., Chao, K.-M., Lo, C.-C., and Wang, Y.-S. Using ontologies to perform threat analysis and develop defensive strategies for mobile security. *Information Technology and Management* (2010), 1–25.