A Context-Aware Malware Detection Based on Low-Level Hardware Indicators as a Last Line of Defense

Alireza Sadighian*, Jean-Marc Robert*, Saeed Sarencheh† and Souradeep Basu‡
*Département de génie logiciel et des TI
École de technologie supérieure, Montréal, Canada
Email: alireza.sadighian.1@ens.etsmtl.ca, jean-marc.robert@etsmtl.ca
†Concordia Institute for Information Systems Engineering (CIISE)
Concordia University, Montréal, Canada
Email: s_saren@encs.concordia.ca
‡School of Computer Science, McGill University, Montréal, Canada
Email: souradeep.basu@mail.mcgill.ca

***Abstract*—Malware detection is a very challenging task. Over the years, numerous approaches have been proposed: signature-based, anomaly-based, application-based, host-based and network-based solutions. One avenue that has been less considered is detecting malware by monitoring of low-level resources consumption (e.g., CPU, memory, network bandwidth, etc.). This can be considered as a last-line of defense. When everything else has failed, the monitoring of resources consumption may detect abnormal behaviors in realtime. This paper presents a context-aware malware detection approach that use semi-supervised machine learning and time-series analysis techniques in order to inspect the impact of ongoing events on the low-level indicators. In order to improve the systems automation and adaptability with various contexts, we have designed a context ontology that facilitates information representation, storage and retrieval. The proposed malware detection approach is complementary to the current malware detectors.**

Keywords–Malware Detection; Low-level Indicators; Context-Aware; Machine Learning; Time-Series Analysis; Ontologies.

I. INTRODUCTION

Today, the emergence of complex heterogeneous infrastructures has led to the evolution of various applications, services, and systems within these computer networks. At the same time, there is an increasing trend of malware exploiting the vulnerabilities of these infrastructures. Hence, technologies and defensive systems aiming to support the efforts of Information Technology (IT) personnel to improve the reliability of their organizations IT assets (i.e., network infrastructure and computer systems) continue to be paramount.

Anomaly-based and signature-based malware detection systems are among the most popular front line tools to protect network infrastructures against malicious attackers. Various approaches have been proposed in the past two decades and commercial off-the-shelf (COTS) malware detection products have found their way into Security Operations Centers (SOC) of most organizations. Nonetheless, the usefulness of these solutions has remained relatively limited due to two main factors: their inability to detect new types of malware (for which new detection rules or training data are unavailable) or simply their high rate of false negative detection and their often very high rate of false positive detection. Due to the increasing prevalence of complex multi-pronged malware, the necessity for organizations to deploy reliable defense systems is undeniable. This is especially important with respect to targeted malware that tries to avoid detection by conventional security products.

One of the essential shortcomings of existing malware detection approaches is that they mostly inspect events on the higher layers of multilayered software or network architectures. Due to the increasing use of metamorphic and polymorphic malware [1], dynamic anomaly-based detection techniques that concentrate on the execution layer or hardware layer are needed more than ever before. The main reason is that attackers do not have control over low-level hardware indicators as they have over higher level features. For example, it is easier for attackers to modify system calls or access control rules than the cache hit rate or the CPU usage rate. As shown in some of the recent works [2], malware events can be differentiated from normal events via their impacts on the low-level feature spaces, such as hardware events collected by performance counters on modern CPUs. Such features have been called sub-semantic because they do not rely on a semantic model of the monitored programs. We believe that sub-semantic features or hardware low-level indicators, such as CPU usage, CPU temperature, memory usage, etc., can be very useful to identify anomalous events in a real-time mode.

Malware detection systems mostly perform offline event analysis. Usually, a dataset of captured events is prepared as an input for these systems to be analyzed. Moreover, they are not easily adaptable with various contexts because a time-consuming configuration process is required. One solution is to propose a real-time, dynamic and highly adaptable malware detection system using ontologies and ontological engineering tools to represent the relevant information [3]. Ontologies provide powerful knowledge representations of the information structure in an unified format [4].

The work presented in this paper strives to address the problems described above, and provide a comprehensive solution to improve the effectiveness of malware detection approaches in real environments. For this purpose, we present a context-aware real-time malware detection approach that relies on ontologies and ontology description logic to accomplish its goals:

1) Analyze impacts on several heterogeneous low-level hardware indicators
2) Identify anomalies using semi-supervised machine learning and time-series analysis techniques

 ISBN: 978-1-61208-582-1

Such a system can be seen as the last line of defense. Whenever the higher level detection mechanisms fail to detect abnormal behaviors, our proposed hardware level system may have the last chance to catch them.

The paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we present our proposed anomaly detection approach in detail. We demonstrate in Section 4 the effectiveness of our proposed approach by describing a reference implementation and applying it to the analysis of two different case studies. We conclude in Section 5 with some insights for future research.

II. Related Work

Anomaly detection, and more specifically, malware detection is one of the main challenges in computer security. A summary of recent studies in anomaly and malware detection is presented in this section.

Khasawneh *et al.* [2] proposed a dynamic malware detection approach based on low-level features, mainly opcodes, to improve the work of Ozsoy *et al.* [5]. In this work, they use a learning approach to perform an online detection and improve detection accuracy. In a way, signatures of the opcodes and similarity graphs of opcode sequences can be considered as low-level features [6] [7] [8]. Abbasi *et al.* [9] considered processor temperature and power consumption as low-level indicators to detect malicious activities in embedded systems. They use a K-means technique to cluster sequences of actions done by processes. Tang *et al.* [10] proposed an unsupervised anomaly-based malware detection using low-level architectural and micro-architectural features available from hardware performance counters.

Detecting anomalies with time-series and temporal sequences has been studied by several researchers [11] [12] [13]. Laptev *et al.* [11] proposed Extendable Generic Anomaly Detection System (EGADS), an automated anomaly detection system based on time-series. They try to detect three classes of anomalies: outliers, change points and anomalous time-series. Chandola *et al.* [12] studied sequence anomaly detection from different perspectives. The authors believe that sequence anomaly detection can be useful for various purposes, such as OS system call analysis, biological sequences analysis (e.g., DNA sequences), and analyzing navigational click sequences from web sites. Lane and Brodley [13] proposed an anomaly detection approach based on Instance-Based Learning (IBL) techniques wherein they transform temporal sequences of discrete, unordered observations into a metric space via a similarity measure that encodes intra-attribute dependencies.

Machine learning techniques, including supervised, semi-supervised and unsupervised techniques, have been widely employed within various anomaly and intrusion detection approaches [14]. Farid *et al.* [15], proposed a learning algorithm for adaptive Network Intrusion Detection Systems (NIDS) based on Naive Bayes and decision trees. Wang *et al.* [16], using feed forward Backward Propagation (BP) neural networks, proposed an intrusion detection approach based on workflow feature definition. Workflows allow to define new attack sequences to assist BP neural networks in order to detect new attack types. Teng *et al.* [17], proposed a cooperative intrusion detection approach using fuzzy Support Vector Machines (SVM), which consists of three detection agents for the Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP) connections.

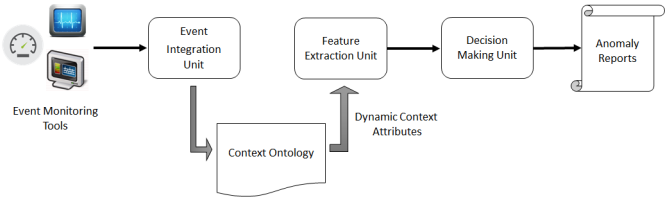

Figure 1. The proposed malware detection framework

In summary, most of these works concentrate on limited aspects of a comprehensive malware detection procedure, such as high-level behavior analysis, a very limited hardware-level low-level indicator analysis, or offline event analysis. However, none of them intends to provide a generic solution dynamically analyzing malware impacts on the normal behavior of the underlying system or network. Motivated by these shortcomings, we propose a context-aware anomaly-based malware detection approach that analyzes dynamically the impact of any event on the hardware-level of the underlying system.

III. The Proposed Malware Detection Approach

In this section, we give a high-level overview of our context-aware malware detection framework as illustrated in Figure 1, which takes full advantage of dynamic events happening within a specific environment. In the first step, the event-integration unit gathers and normalizes the information provided by the different monitoring tools. These tools installed in different architectural layers of the underlying environment provide a wide range of contextual information which can be used to significantly improve the accuracy of the final decisions of the detection framework.

In the second step, the context ontology is populated with the information collected by the monitoring tools. This ontology facilitates traversing (drilling down and rolling up) various levels of the underlying environment to extract very generic or very specific information. It provides dynamic information on the underlying environment for real-time analysis. This information is *normalized* in some way to ease the analysis.

In the third step, the feature extraction unit queries the context ontology to retrieve useful information required for its analysis. It retrieves top meaningful features that provide useful data for a sophisticated malware detection system.

The last step consists of detecting anomalous events happening in the underlying environment. For this purpose, semi-supervised machine learning and time-series analysis techniques are employed in this phase.

A. Event Monitoring Tools

In order to track the impact of any event happening within a network, we need monitoring tools to oversee the behavior of the main components based on various low-level indicators, such as CPU usages, memory usage, disk usage, incoming/outgoing traffic rates, etc. Some of the main components that attackers usually try to bypass or compromise are: network firewall, web server, email server, Intrusion Detection System (IDS), etc. Hence, monitoring the behavior of low-level

 ISBN: 978-1-61208-582-1

TABLE I. AN EXAMPLE LIST OF LOW-LEVEL INDICATORS

CPU	Utilization Time	IOWaits	Network	Incoming	traffic on each interface
		Nice			dropped packets
		Softirq			errors on each interface
		User			Packet Loss
		System		Outgoing	traffic on each interface
		Steal			dropped packets
		Idle			errors on each interface
	Load				Packet Loss
	Number of interrupts				
	Context switches/second				
Memory	Physical Memory	Available	Disk	Free space for each filesystem	
		Total		Used space for each filesystem	
	Swap	Available		Total space for each filesystem	
		Total		Free inodes for each filesystem	
OS	Max number of opened files		System	Up-time	
	Number of processes			Local time	
	Max number of processes			Boot-time	
	Host reachability using ICMP			Number of logged in users	
	Server configured domain			Host location	
	Check name resolution				

TABLE II. THE LIST OF ATTRIBUTES OF THE CONTEXT ONTOLOGY CLASSES

Organization	Network	Host	User	OS	Application
ID Product Client Location Network	Topology Protocol Address Range Firewall Switch Host User #hosts #subnets #switches	Name IP/MAC Address Role Location User Service OS Application CPU / Memory #user / #OS	ID Role Location Access rights	Platform Type SPVersion Version	Name Version
	Traffic Type	Memory/CPU Usage Disc Usage Open ports Started Apps Current Users Connections Status Application CPU/Memory Sent/Received Bytes	Per User Per Host Per Net	Syscall Table Process	Started State

indicators in these components provides useful information to detect various malware.

Table I lists the low-level indicators that we monitor. Thus, anomaly-based monitoring tools should take advantage to monitor the behavior of such indicators. They would look for any significant changes, such as an abrupt increase or decrease over a short period of time (a burst).

B. Event Integration Unit

In general, monitoring tools provide reports in various formats that might not be natively interpretable by the context ontology and the malware detection engines. Hence, it is necessary to preprocess these reports and export them in a format that is understandable by both engines. In production environments, this would be done by specific drivers that would match monitoring fields with class attributes at the appropriate abstraction level. In the proposed framework, the event integration unit converts the collected events from monitoring tools to a unified format which can be understood by the next units. The other major tasks of the event integration unit are as follows:

- The monitoring tools may generate attributes in different types (string, integer, etc.). The event integration unit transforms all the received information (attribute values) to a unified type for the ontology engine.
- Some of the monitoring tools may not support particular class attributes. The event integration unit completes the missing data and attributes.
- The event integration unit removes noises and meaningless values in the collected data from monitoring tools.

Once the integration process has been completed by the event integration unit, the context ontology is populated using the normalized information.

C. Context Ontology

Ontologies provide a powerful knowledge representation in a unified format which is understandable by both machines and humans [4]. Ontologies allow the use of reasoning logic formalisms that can be used to retrieve information in a generic structure-agnostic fashion. We use these formalisms to design our real-time malware detection algorithms. Our main objective is to detect complex and challenging malware that bypasses current security solutions. The use of ontologies and ontology description logic enables us to fully automate the dynamic contextual information retrieval that is typically done manually by the analysts.

In our malware detection framework, we have designed a context ontology easily adaptable to various environments, indicating its flexibility and power of abstraction. Additionally, it is highly extensible to include more contextual classes and attributes depending on the level of abstraction.

The context ontology is populated using the information integrated by the event integration unit. Figure 2 illustrates our designed context ontology, which has been implemented using the popular open-source ontology editor Protégé (as shown in Figure 11 of Appendix C). The context ontology includes a Context base class and User, Host, Network and Service associated classes with their corresponding attributes. Each of these classes has both static (above the line) and dynamic (below the line) attributes (as listed in Table II). These attributes are provided either by network fingerprinting tools or network administrators.

In order to navigate among various levels of class hierarchies within the context ontology, we use the following two operators in the form of a set of logic rules expressed in Semantic Query-Enhanced Web Rule Language (SQWRL) [18]:

- **Drill-Down** allows to navigate among levels of data ranging from the most summarized to the most detailed concepts.
- **Roll-Up** allows to navigate among levels of data ranging from the most detailed to the most summarized concepts.

D. Feature Selection Unit

Once the context ontology has been populated with the dynamic events of the underlying monitored system, this information can be used to detect potential anomalous events.

The first step is therefore to query the context ontology to extract dynamic information on the environment and to prepare them for analysis. We use a set of logic rules expressed in

 ISBN: 978-1-61208-582-1

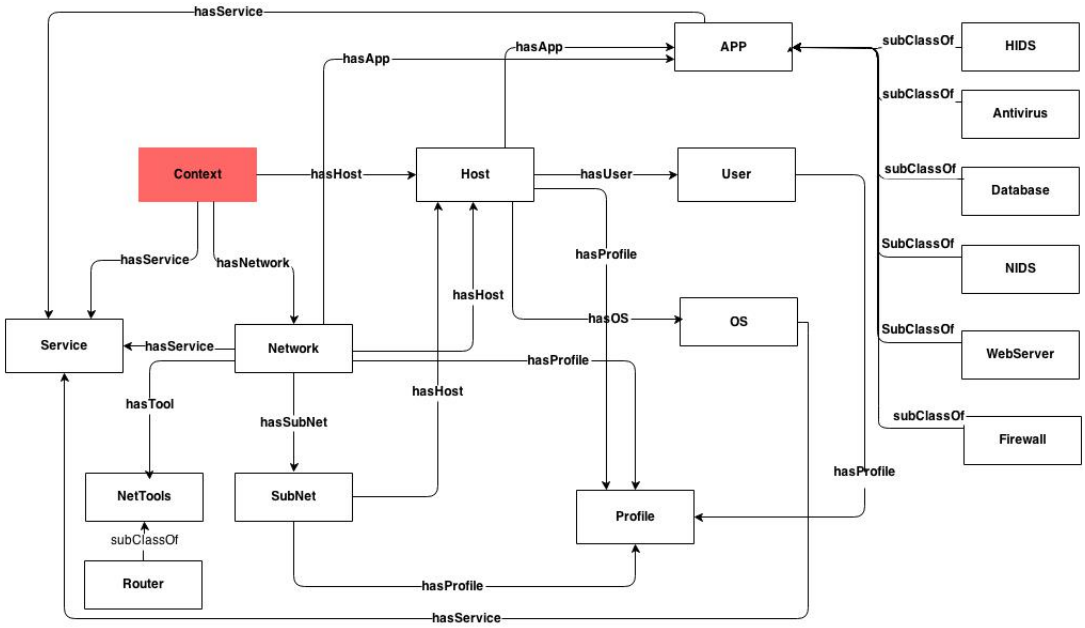

Figure 2. Class diagram relationship of the context ontology

SQWRL to query the context ontology. These rules facilitate the traversing of the ontology class hierarchy to retrieve the important attributes for the anomaly detection algorithms. Some examples are given in the appendix. These queries extract events, which are represented as follows: e_i is a feature vector $x_i = (x_{i,1}, x_{i,2}, \cdots, x_{i,m})$.

Several attributes of the context ontology can be used as input features for machine learning techniques to detect anomalous events. Using various feature analysis algorithms [19], the most useful features can be selected for the analysis by the decision-making unit. Principal Component Analysis (PCA) [19] and parallel coordinates [20] have been used for this paper. PCA is a statistical procedure that transforms a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. On the other hand, parallel coordinates is a way of visualizing high-dimensional geometry and analyzing multivariate data. It eases feature selections by analysts.

In order to have a real-time malware detection system, the feature selection unit consists of a number of pre-defined triggers for the most critical features (e.g., CPU usage, memory usage). When one of these triggers is activated, it starts extracting contextual information from the context ontology as machine learning features, and selects those features providing meaningful information.

E. Decision Making Unit

In this section, we provide a detailed picture of the proposed malware detection approach. Two different approaches are used: machine learning techniques and time-series analysis. Semi-supervised machine learning techniques [21] (e.g., One-Class Support Vector Machine (OC-SVM)) have been chosen because the cost associated with the event labeling process in supervised machine learning techniques is significantly high, whereas acquisition of unlabeled or partly labeled data is significantly inexpensive. Time-series analysis techniques [22] (e.g., Cumulative Sum (CUSUM) [23]) try to extract meaningful statistics and internal structure of the input data. These two techniques complement each other considering that OC-SVM does not take into account internal correlation of events, while time-series analysis accounts for the fact that data points taken over time may have an internal structure, such as auto-correlation, trend or seasonal variation. Time series reflect also the stochastic nature of events over time. Hence, data may be skewed, with fluctuating mean and variation, non-normally distributed, and not randomly sampled. The pseudo-code for the proposed malware detection approach is presented in Figure 10 (Appendix A).

1) Detecting Anomalous Events Using OC-SVM: The One-Class Support Vector Machine (OC-SVM) has been used since it can be trained with only normal events. OC-SVM can be viewed as a regular two-class SVM where all the training data lies in the first class, and the origin is taken as the only member of the second class. Then, in the testing phase, any abnormal event is considered as an outlier in theory. This removes the need to gather attacks or abnormal traffic. Hence, the main idea is to classify the training data as positive, and classify testing data as negative only if it is sufficiently different from the training data.

A One-Class SVM is a linear classifier in a multi-dimensional feature space [21]. It maps a hyper-sphere to the input data in order to separate normal events from the origin. These points lying inside the hyper-sphere are classified as outliers. This can be formulated as an optimization problem as follows:

$$\begin{aligned} &\min_{R,b,\xi} R^2 + \frac{1}{vn}\sum_{i=1}^{n}\xi_i \\ &\text{subject to} : \left(\|\phi(x_i) - b\|^2 \leq R^2 - \xi_i \quad \text{and} \quad \xi_i \geq 0\right) \end{aligned} \tag{1}$$

where, b and R are the center and radius of the hyper-sphere, and ξ is the slack variable. When v is small, we try to put

more data into the ball. On the other hand, when v is larger, we try to squeeze the size of the ball. This optimization can be solved by Lagrangian multipliers.

2) Malware Detection Based on Time-Series Analysis: Detecting anomalous event sequences is one of the most important requirements of any malware detection system to prevent potential disasters. Sometimes, a single event does not sound anomalous, whereas, a sequence of such events can represent malicious behavior. Sending a few emails per hour sounds normal for a trusted computer system. However, sending thousands of emails per hour may demonstrate that the system has been compromised by spammers.

A time-series consists of a sequence of events obtained over repeated measurements of time [24]. An anomalous time-series is defined as a time-series whose average deviation from the other time-series is significant. In order to detect anomalous time-series, we use the CUSUM technique.

CUSUM is a standard sequential analysis technique used for online changepoint detection [23]. In the following, we describe the procedure of calculating CUSUM for a sequence of events $(x_0...x_n)$. For each event, a probability density function (PDF) $p(x_i, \theta)$ depending on a deterministic parameter θ is defined. In the case of a changepoint at time t_c, we define $\theta = \theta_0$ before t_c and $\theta = \theta_1$ after t_c. CUSUM uses a likelihood ratio test as its changepoint detection theory. Thus, the $instantaneous \quad log - likelihood \quad ratio$ at time i is defined as follows:

$$s[i] = L_x[i, i] = \ln\left(\frac{p(x_i, \theta_1)}{p(x_i, \theta_0)}\right) \tag{2}$$

and CUSUM from 0 to k: $S[k] = \sum_{i=0}^{k} s[i]$. Accordingly, the decision function $G_x[k]$ and change time estimate $\widehat{i}$ will be defined as follows:

$$G_x[k] = S[k] - \min_{1 \le i \le k} S[i-1] \tag{3}$$

$$\widehat{i} = arg \min_{1 \le i \le k} S[i-1] \tag{4}$$

Equation (4) shows that the change time estimate is the time following the current minimum of the cumulative sum. The value of decision function $G_x[k]$ is zero before the changepoint and increasing afterwards. When the value of $G_x[k]$ exceeds a certain threshold, an anomalous event or event sequence is detected.

Thus, our approach to detect anomalous event sequences consists of the following phases:

1) Concentrate on the training dataset to find the thresholds describing the normal behaviors of the system, such as the decision interval and the shift decision. We first calculate CUSUM of the training dataset. As a result, we find the threshold interval of the CUSUM approach for the training dataset.
2) Analyze the testing dataset to list potential anomalies. For this purpose, first, the CUSUM of the testing dataset is calculated. Next, all the events or event sequences having CUSUM greater than the highest threshold or less than the lowest threshold will be reported as anomalous event sequences.

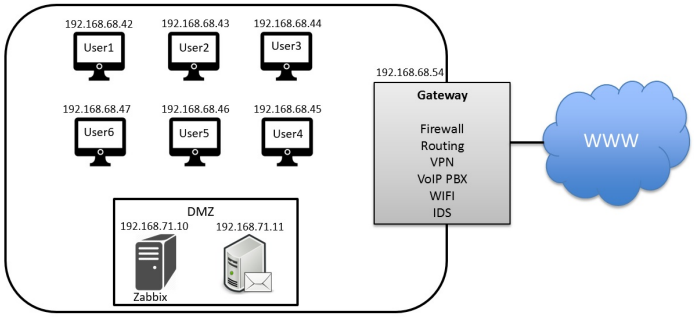

Figure 3. The experiments test-bed

IV. Implementation and Evaluation

In order to illustrate and validate our approach, we have developed a reference implementation using a collection of tools including network monitoring tools, ontology representation and reasoning tools, as well as machine learning and time-series platforms.

We used this collection of tools to conduct two experiments in a real network as our field test-bed. Figure 3 illustrates our test-bed for the experiments. The normal users of this network are mostly programmers and researchers. A Zabbix server is used to monitor the behavior of the low-level indicators (CPU usage, memory usage, CPU temperature, network traffic, etc.) of the critical components, such as the network gateway and the email server, during the experiments. The network gateway (192.168.68.54) provides several services, such as network firewall, routing, Virtual Private Network (VPN), Voice over IP Private Branch Exchange (VoIP PBX), WIFI and NIDS.

In this section, we describe the reference implementation of our malware detection framework and present two case studies to demonstrate how it can detect various types of anomalous events happening in a real computer network.

A. Reference Implementation

Our event monitoring solution relies on the open source monitoring software Zabbix [25], and our event integration tool relies on the agent-less universal Security Information Management System (SIEM) Prelude [26].

As mentioned earlier, the Protégé ontology editor and knowledge acquisition system [27] has been used to design and implement our context ontology using the Ontology Web Language Description Logic (OWL-DL). The context ontology is instantiated with the normalized information coming from Prelude. Furthermore, the Pellet plug-in [28] is used as a reasoner for OWL-DL, and SQWRL to query the ontologies for various purposes.

For feature selection and machine learning-based decision making, we use scikit-learn [29], pyplot NumPy [30] and SciPy [31] Python libraries. Finally, we use CUSUM (a library in R) for time series-based decision making.

B. Case Study 1: Detecting TorrentLocker Ransomware

Our first case study is to detect TorrentLocker ransomware. TorrentLocker is a ransomware that encrypts private data of infected computer systems, and asks users to pay a ransom (usually, in Bitcoins) to re-gain access to their data. Once TorrentLocker infects a system, it encrypts the first two megabytes

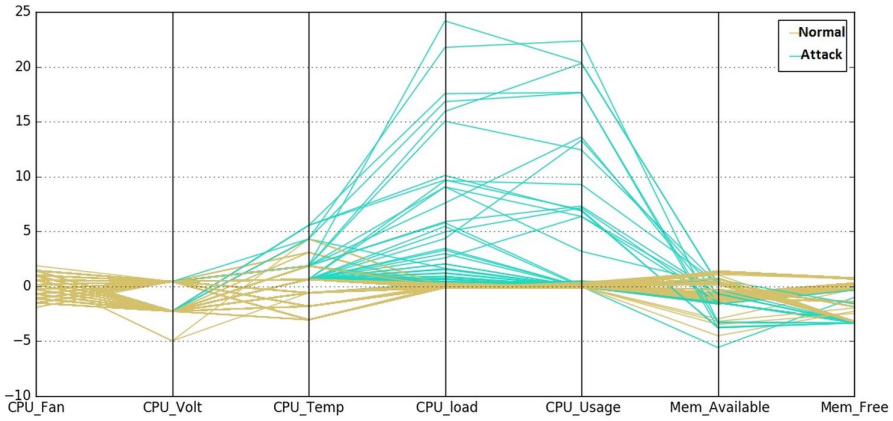

Figure 4. Low-level indicators during the ransomware detection experiment

of all the existing files found on that system. Encrypting partially the files is sufficient to conceal the information and is more efficient for the malware. Unfortunately, currently, antiviruses and intrusion detection systems have difficulties to detect such polymorphic malware.

In this case study, we simulate TorrentLocker behavior wherein once the ransomware infects the network gateway (Figure 3), it starts encrypting all the existing files. For this purpose, in a similar way as TorrentLocker, we launch multiple suspicious processes (multi-threaded Python scripts) accessing a large number of structured files, encrypting them using the AES-256 encryption algorithm in CBC mode, and overwriting them with encrypted files. Our dataset includes 6000 JPEG files (1MB each).

We conducted a one-week (work-days from 10:00 to 17:00) experiment in a real network. During the first five days of the experiment, we captured the normal behavior of the network gateway and prepared the training dataset. The training dataset includes low-level hardware indicators (e.g., CPU usage, memory usage, disk usage, etc.) of the normal events. The last two days was used to test our solution. To simulate TorrentLocker ransomware behavior, we ran the ransomware in several steps (Table III). In each step, we ran the ransomware with different number of threads and files. To discover which low-level indicators have been affected during the experiment, we queried the context ontology (Rule 2 in Appendix B) and visualized them using parallel coordinates. Figure 4 illustrates the extracted features. Each feature has been shown by a separate coordinate. Different colors have been used to visualize normal and abnormal events. Horizontal lines indicates how each event affects the extracted features. As shown, the main features affected during the simulated attack are: CPU Temperature, CPU Usage, CPU Load and Memory Free. As Figure 5 illustrates, CPU Usage is the key feature targeted by the attack.

In order to start analyzing this suspicious event using our proposed approach, we applied PCA algorithm to the extracted feature list to reduce data dimension. As Table IV shows, features like CPU Temp, CPU Load, Memory Available and CPU Usage have been mainly affected during this experiment. In the rest of this section, we explain how the decision-making unit of our malware detection framework employs the final feature list to detect the anomalous activities.

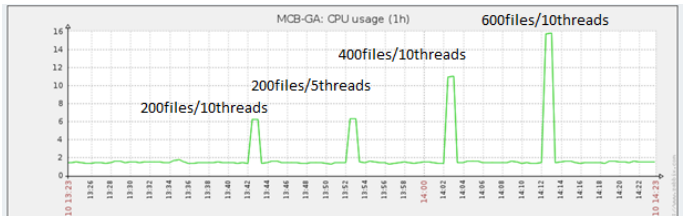

Figure 5. CPU usage during the test-day

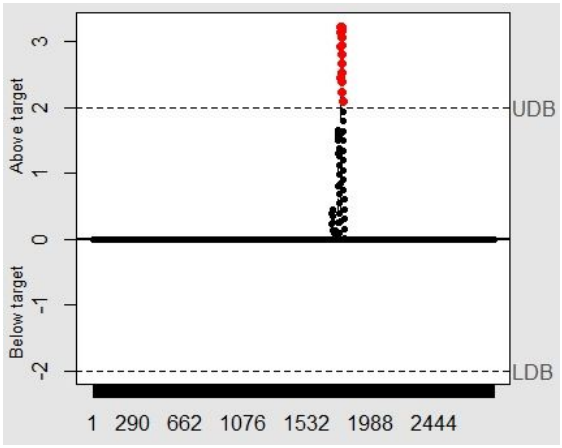

Figure 6. Time-series analysis of test data in Case Study 1

The decision-making unit has been developed based on two main techniques: OC-SVM and time-series analysis. First, OC-SVM was trained using the training dataset. Next, OC-SVM was applied to the test dataset that contains the anomalous event sequences. Table V shows the results. The amount of False Positive (FP), True Positive (TP), False Negative (FN) and True Negative (TN) are given. These statistics can be used to compute the $accuracy = \frac{(TP+TN)}{(TP+TN+FN+FP)}$. Except for Step 6, all the anomalous steps were detected by OC-SVM. Step 6 was not detected because its impact on the low-level indicators is significantly low.

In the second phase, we used the time-series analysis technique to detect anomalous event sequences. Thus, the time-series analysis module was trained using event sequences to learn normal thresholds, such as the decision interval and the

TABLE III. RANSOMWARE EXPERIMENT STEPS

Steps	1	2	3	4	5	6	7	8	9	10
Day-time	F-13:42	F-13:52	F-14:02	F-14:12	F-14:42	F-14:52	S-14:14	S-14:24	S-14:34	S-14:44
File #	200	200	400	600	100	50	2000	2000	4000	6000
Thread #	10	5	10	10	10	10	50	100	100	100

TABLE IV. THE RESULT OF APPLYING PCA TO THE EXTRACTED FEATURES

Attribute	PC 1	PC 2	PC 3
CPU_Fan		-0.258	0.652
CPU_Volt		0.375	-0.342
CPU_Temp	-0.226	-0.207	-0.555
CPU_Load	-0.555	0.231	0.18
CPU_Usage	-0.538	0.21	0.197
Mem_availability	0.158	0.651	
Mem_Free	0.307	0.449	0.28

TABLE V. RESULTS FOR CASE STUDY 1

Alarms	OC-SVM	Time-Series
FP	21	146
TP	46	9
FN	19	56
TN	1474	1349
Accuracy	0.97	0.87

shift. Next, the time-series analysis module was applied to the test dataset. The results (Table V) show that using time-series analysis, we were able to detect only step 10. The main reason is that the number of events generated during the first 9 steps is less than the considered time-series window size. Figure 6 illustrates how time-series analysis processes the data to detect anomalous event sequences.

Consequently, overall, our proposed anomaly detection approach succeeded to detect all the abnormal activities of this experiment except the abnormal activity of Step 6 that very lightly affected the system low-level indicators. This means, our proposed approach was able to detect TorrentLocker ransomware by sacrificing only 100 files of the infected system.

C. Case Study 2: Spamming Bot

We evaluate here our malware detection approach using a spamming bot scenario wherein a compromised machine (192.168.68.45). inside the network sends a massive number of spam emails that significantly affects incoming and outgoing traffic rate within the network gateway. For this purpose, we conducted a three day (work-days from 10:00 to 17:00) experiment in our network. During the first two days, we captured the normal behaviors of the network gateway and prepared the training dataset, which includes low-level hardware indicators (e.g., CPU, memory and disk usage, incoming and outgoing traffic, etc.).

The last day was dedicated to prepare the testing dataset. A bot machine started to send a massive number of spam emails at time periods 14:41-15:11 (100 kb/s), 15:22-15:52 (400 kb/s), 16:03-16:33 (800 kb/s) and 16:43-17:13 (1.1 mb/s). This produced a very large traffic rate on the network gateway which affects a number of low-level indicators.

TABLE VI. RESULTS FOR CASE STUDY 2

Alarms	OC-SVM	Time-Series
FP	19	0
TP	35	186
FN	205	34
TN	1301	1340
Accuracy	0.87	0.98

In order to discover which low-level indicators have been significantly affected during the experiment, we queried the context ontology for a number of features and visualized them using parallel coordinates (Figure 7). As we see, CPU Temperature/Usage/Load and Free Memory are the main features affected during this scenario. Figure 8 illustrates CPU Load and CPU Temperature behaviors during the test-day. Next, we applied the PCA algorithm to the extracted feature list to reduce data dimension. In the following paragraphs, we explain how the decision-making unit of our malware detection framework is able to detect such abnormal activities.

We applied the two phases of the decision making process (same as Case Study 1) to the training and test dataset. Table VI shows the obtained results. Figure 9 illustrates how time-series analysis processes the input data to detect anomalous event sequences. The results indicate that both OC-SVM and time-series analysis module ware able to detect the anomalous events. Consequently, the proposed malware detection approach successfully detected the abnormal activities of this experiment.

As the first phase of the decision making process, first, OC-SVM was trained using the training dataset. The maximum CPU load in the training dataset were 2.19. Next, it was applied to the test dataset that contains four abnormal activities. Table 5 shows the results. The results indicate that OC-SVM detected only one of the abnormal activities (the highest traffic) as its CPU load was higher than maximum CPU load in the training dataset.

In the second phase, first, we trained the time-series analysis module using event sequences of training dataset to learn normal thresholds. The time-series analysis module was applied to the test dataset. The obtained results (Table VI) show that using time-series analysis, we were able to detect three of the anomalous abnormal activities. The first abnormal activity was not detected as its impact on low-level indicators was mostly similar to normal activities. Consequently, the proposed anomaly detection approach successfully detected three abnormal activities of this experiment.

V. CONCLUSIONS

In this paper, we discussed the shortcomings of malware detection systems (e.g., inability to detect new types of attacks and the often very high rate of false positives), and proposed a new context-aware anomaly-based malware detection approach

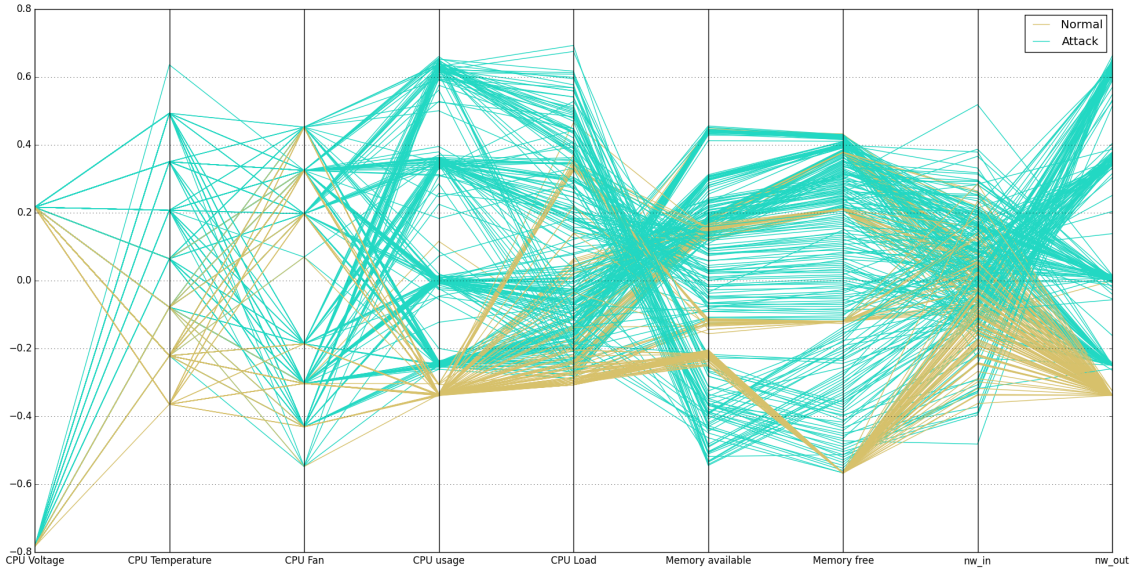

Figure 7. Low-level indicators behavior during the spamming bot experiment

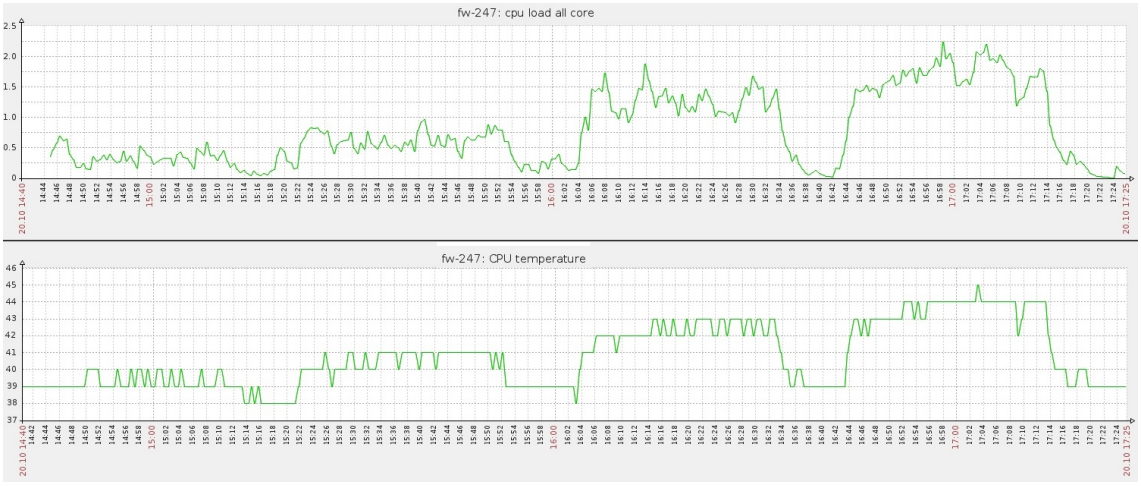

Figure 8. CPU load and CPU temperature during the test-day

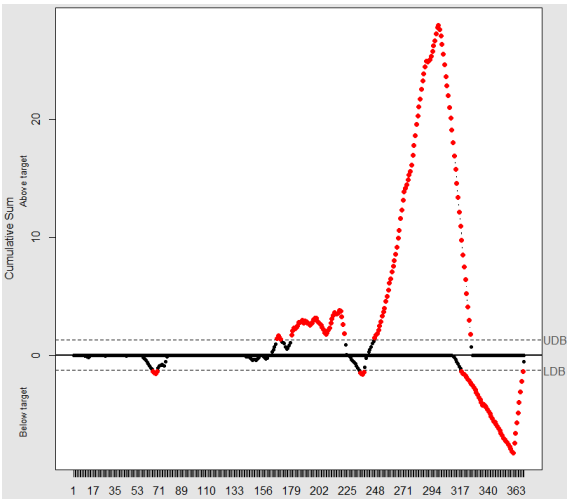

Figure 9. Time-series analysis of test data in Case Study 2

based on low level sensors as a last-line of defense to overcome these shortcomings. If malicious attackers may be able to deactivate firewall or IDS, they cannot alter the low level sensors.

The main idea of our approach is to detect any anomaly at the hardware layer by verifying legitimacy or maliciousness of an event or an event sequence based on the impacts that it enforces to the underlying monitoring low-level indicators (e.g., CPU/memory usage, network traffic rate, etc.). For this purpose, several monitoring tools are employed to collect and analyze low-level indicators behavior in a real-time mode. To do so in a manner that can be automated, but that yet can be easily extended to new concepts (richer concepts of context), we used ontologies and ontological engineering tools to represent knowledge and information about contextual information using the Ontology Web Language (OWL). We proposed the ontology for contextual information accordingly, considering the capability to import both explicit contextual information from Configuration Management Systems (CMS)

 ISBN: 978-1-61208-582-1

or implicit contextual information obtained from users and system profiling techniques. In order to verify legitimacy or maliciousness of ongoing events, we used semi-supervised machine learning and time-series analysis techniques that complement each other to identify both anomalous events and sequences.

To illustrate our approach, we implemented our new approach on two distinct case studies (i.e., remote code execution attack scenario and spamming bot scenario) designed based on current challenging malware and attack scenarios, we successfully evaluated the proposed anomaly detection approach in a real network environment. The results show that our proposed approach can successfully detect abnormal behaviors at a very low system level.

By 1) collecting more and more normal events from the underlying context in order to appropriately train and adjust the OC-SVM and time-series analysis module, and 2) adding more low-level indicators of the underlying context to the context ontology, both false positive and false negative rates will be significantly reduced. Hence, the reliability of the proposed malware detection approach will be improved. Consequently, our approach can appropriately complement existing malware detection approaches that mostly inspect events on the higher layers of multilayered software or network architectures without taking into account the execution layer or hardware layer.

As our future work, we intend to populate the context ontology with more sophisticated context models, populated with network fingerprinting and profiling tools. We also plan to evaluate our proposed anomaly detection approach against other complex attacks in order to reliably gauge its performance and effectiveness in real-life situations.

ACKNOWLEDGMENT

We would like to thank Groupe Access which allows us to develop our low-level sensors in their network. This research was sponsored in part by Mitacs Canada and Groupe Access Inc.

REFERENCES

[1] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in 2010 International conference on broadband, wireless computing, communication and applications. IEEE, 2010, pp. 297–300.

[2] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in Research in Attacks, Intrusions, and Defenses. Springer, 2015, pp. 3–25.

[3] A. Sadighian, J. M. Fernandez, A. Lemay, and S. T. Zargar, "Ontids: A highly flexible context-aware and ontology-based alert correlation framework," in Foundations and Practice of Security. Springer, 2014, pp. 161–177.

[4] A. Gomez-Perez, M. Fernández-López, and O. Corcho, Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. Springer Science & Business Media, 2006.

[5] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on. IEEE, 2015, pp. 651–661.

[6] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection," Journal in Computer Virology, vol. 8, no. 1-2, 2012, pp. 37–52.

[7] I. Santos et al., "Idea: Opcode-sequence-based malware detection," in Engineering Secure Software and Systems. Springer, 2010, pp. 35–43.

[8] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2013, pp. 41–61.

[9] Z. Abbasi, M. Kargahi, and M. Mohaqeqi, "Anomaly detection in embedded systems using simultaneous power and temperature monitoring," in Information Security and Cryptology (ISCISC), 2014 11th International ISC Conference on. IEEE, 2014, pp. 115–119.

[10] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in Research in Attacks, Intrusions and Defenses. Springer, 2014, pp. 109–129.

[11] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015, pp. 1939–1947.

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," Knowledge and Data Engineering, IEEE Transactions on, vol. 24, no. 5, 2012, pp. 823–839.

[13] T. Lane and C. E. Brodley, "Temporal sequence learning and data reduction for anomaly detection," ACM Transactions on Information and System Security (TISSEC), vol. 2, no. 3, 1999, pp. 295–331.

[14] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM computing surveys (CSUR), vol. 41, no. 3, 2009, pp. 1–58.

[15] D. M. Farid, N. Harbi, and M. Z. Rahman, "Combining naive bayes and decision tree for adaptive intrusion detection," arXiv preprint arXiv:1005.4496, 2010.

[16] Y. Wang, D. Gu, W. Li, H. Li, and J. Li, "Network intrusion detection with workflow feature definition using bp neural network," in Advances in Neural Networks–ISNN 2009. Springer, 2009, pp. 60–67.

[17] S. Teng, H. Du, N. Wu, W. Zhang, and J. Su, "A cooperative network intrusion detection based on fuzzy svms," Journal of Networks, vol. 5, no. 4, 2010, pp. 475–483.

[18] M. J. O'Connor and A. K. Das, "Sqwrl: A query language for owl." in OWLED, vol. 529, 2009.

[19] I. Jolliffe, Principal component analysis. Wiley Online Library, 2002.

[20] A. Inselberg, Parallel coordinates. Springer, 2009.

[21] Y. Chen, X. S. Zhou, and T. S. Huang, "One-class svm for learning in image retrieval," in Image Processing, 2001. Proceedings. 2001 International Conference on, vol. 1. IEEE, 2001, pp. 34–37.

[22] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, Time series analysis: forecasting and control. John Wiley & Sons, 2015.

[23] P. Granjon, "The cusum algorithm-a small review," 2014.

[24] H. Madsen, Time series analysis. CRC Press, 2007.

[25] L. Zabbix, "Zabbix," The Enterprise-class Monitoring Solution for Everyone2016, Available from http://www. zabbix. com, 2015.

[26] K. Zaraska, "Prelude ids: current state and development perspectives," URL http://www. prelude-ids. org/download/misc/pingwinaria/2003/paper. pdf, 2003.

[27] M. A. Musen, "Protégé ontology editor," Encyclopedia of Systems Biology, 2013, pp. 1763–1765.

[28] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," Web Semantics: science, services and agents on the World Wide Web, vol. 5, no. 2, 2007, pp. 51–53.

[29] F. Pedregosa et al., "Scikit-learn: Machine learning in python," The Journal of Machine Learning Research, vol. 12, 2011, pp. 2825–2830.

[30] W. McKinney, Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.", 2012.

[31] E. Jones, T. Oliphant, and P. Peterson, "{SciPy}: open source scientific tools for {Python}," 2014.

 ISBN: 978-1-61208-582-1

APPENDIX

A. Malware Detection Pseudocode

INPUT: Train-Event-List, Test-Event-List.
OUTPUT: Malware-Report
BEGIN

{Training}
for all $e \in Train - Event - List$ **do**
  $OC - SVM(e)$
end for

while (Train-Event-List is not empty) **do**
  $w \leftarrow createwind(Train - Event - List)$
  $CUSUMList \leftarrow CUSUM(w)$
  $h \leftarrow max(CUSUMList)$
end while

{Testing}
for all $e \in Test - Event - List$ **do**
  $EventClass \leftarrow OC - SVM(e)$
  if $EvenetClass\ is\ attack$ **then**
    $attack - list1 \leftarrow e$
  end if
end for

while (Test-Event-List is not empty) **do**
  $w \leftarrow createwind(Test - Event - List)$
  if $CUSUM(w) > h$ **then**
    $attack - list2 \leftarrow w$
  end if
end while

$Malware - Report \leftarrow attack - list1 \bigvee attack - list2$

END

Figure 10. The Proposed Malware Detection Pseudocode

B. Rule Examples

Rule 1 extracts all the events in the Web Server (192.168.71.247) having CPU usage higher than 60%. The outcome event list can be analyzed in order to discover any potential cause of abnormal CPU usage.

Rule 1:

$$\texttt{Event}(?e_1) \wedge \texttt{Host}(?h_1) \wedge \texttt{hasAddress}(?h_1, \texttt{"192.168.71.247"}) \wedge \texttt{hasSource}(?e_1, ?h_1) \wedge \texttt{hasCPUUsage}(?e_1, ?\texttt{cpuusage}) \wedge \texttt{greaterThanOrEqual}(?\texttt{cpuusage}, \texttt{"60\%"}) \longrightarrow \texttt{sqwrl:select}(?e_1)$$

Rule 2 extracts values of a set of low-level hardware indicators (i.e. Timestamp, CPU Usage, CPU Voltage, CPU Temperature, CPU Fan, CPU Load, Network Input, Network Output, Available Memory and Free Memory) in the Network Gateway (192.168.68.54) to be analyzed in order to discover major affected features.

Rule 2

$$\texttt{Host}(?h) \wedge \texttt{hasAddress}(?h, \texttt{"192.168.68.54"}) \wedge \texttt{hasTimeStamp}(?h, ?\texttt{timestamp}) \wedge \texttt{hasCPUUsage}(?h, ?\texttt{cpuusage}) \wedge \texttt{hasCPUVoltage}(?h, ?\texttt{cpuvoltage}) \wedge \texttt{hasCPUTemperature}(?h, ?\texttt{cputemperature}) \wedge \texttt{hasCPUFan}(?h, ?\texttt{cpufan}) \wedge \texttt{hasCPULoad}(?h, ?\texttt{cpuload}) \wedge \texttt{hasNetworkInput}(?h, ?\texttt{networkinput}) \wedge \texttt{hasNetworkOutput}(?h, ?\texttt{networkoutput}) \wedge \texttt{hasMemoryAvailable}(?h, ?\texttt{memoryavailable}) \wedge \texttt{hasMemoryFree}(?h, ?\texttt{memoryfree}) \longrightarrow \texttt{sqwrl:select}(?\texttt{timestamp}, ?\texttt{cpuusage}, ?\texttt{cpuvoltage}, ?\texttt{cputemperature}, ?\texttt{cpufan}, ?\texttt{cpuload}, ?\texttt{networkinput}, ?\texttt{networkoutput}, ?\texttt{memoryavailable}, ?\texttt{memoryfree})$$

C. Ontology Implementation

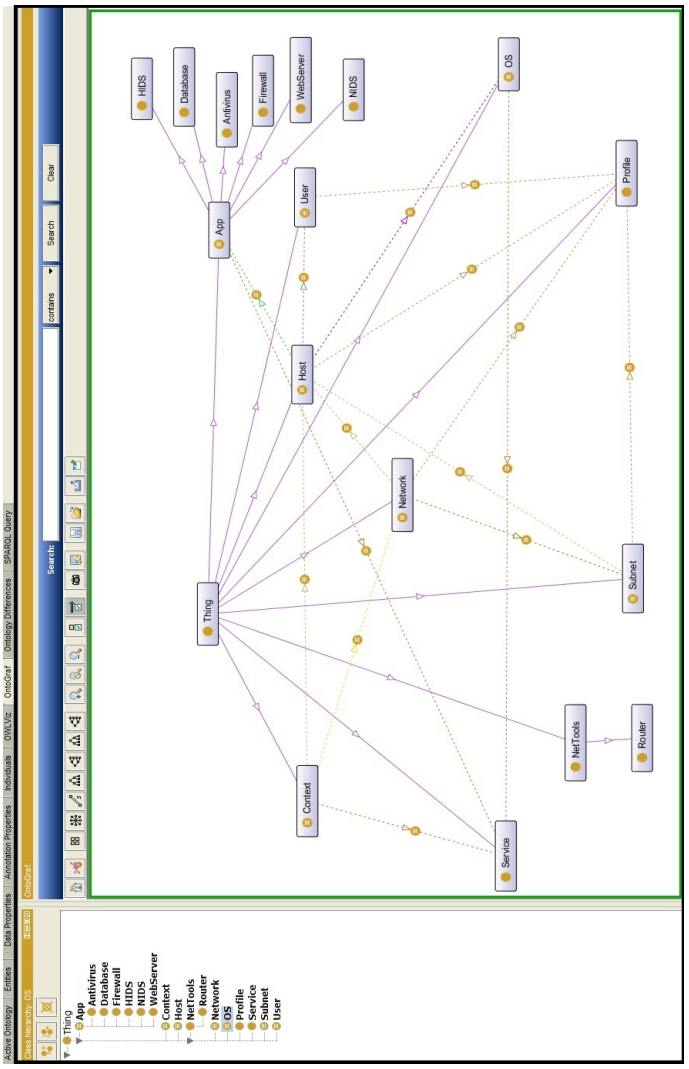

Figure 11. Implementation of the context ontology using Protégé