IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 14, 2019, accepted January 6, 2020, date of publication January 13, 2020, date of current version January 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966321

MACoMal: A Multi-Agent Based Collaborative Mechanism for Anti-Malware Assistance

MOHAMED BELAOUED[1], ABDELOUAHID DERHAB[2], SMAINE MAZOUZI[3], AND FARRUKH ASLAM KHAN[2], (Senior Member, IEEE)

[1]LIRE Laboratory, Software Technologies and Information Systems Department, University of Constantine 2, Constantine 25001, Algeria
[2]Center of Excellence in Information Assurance (COEIA), King Saud University, Riyadh 11451, Saudi Arabia
[3]Department of Computer Science, Université 20 Août 1955-Skikda, Skikda 21000, Algeria

Corresponding author: Mohamed Belaoued (belaoued.mohamed@gmail.com)

This work was supported by the Deanship of Scientific Research at King Saud University, through Research Group no. RG-1439-021.

ABSTRACT Anti-malware tools remain the primary line of defense against malicious software. There is a wide variety of commercial anti-malware tools in the IT security market. However, no single tool is able to provide a full protection against the overwhelming number of daily released malware. Hence, collaboration among malware detection tools is of paramount importance. In this paper, we propose MACoMal, a multi-agent based decision mechanism, which assists heterogeneous anti-malware tools to collaborate with each other in order to reach a consensual decision about the maliciousness of a suspicious file. MACoMal consists of two main elements: (1) an executable file identification model, and (2) a collaborative decision-making scheme. MACoMal is analyzed with respect to network connectivity and global decision correctness. By leveraging a multi-agent simulation tool and a set of real malware samples, we present a simulation methodology to assess its effectiveness and efficiency. Experimental results show that MACoMal is able to immunize a network against a malware threat within a time that ranges from a few seconds to a few minutes after the threat detection.

INDEX TERMS Malware, anti-malware assistance, multi-agent systems, modelling, analysis, simulation, collaboration.

I. INTRODUCTION

The term Malware refers to *a group of software designed to penetrate or damage a computer system without the owner's knowledge* [1]. Malware have various forms (i.e., viruses, trojans, worms, etc.), and still account for the majority of cyber attacks [2]–[6]. According to a report from Symantec [7], more than 246 million new malware were discovered in 2018, which represents approximately 674 thousand new malware per day. In order to defend against this overwhelming threat, a wide variety of anti-malware tools are provided by the IT security vendors. However, no single tool can provide a full protection against malware threats [8]–[11]. This is mainly due to the following challenges:

1) The average time required for an anti-malware tool to detect new threats (zero-day malware) can range from few hours to several weeks [8]–[10] during which the system remains vulnerable to the overwhelming number of malware released everyday.
2) Each anti-malware tool uses a different method for malware detection, and generates signatures based on its own collected malware samples. Therefore, a tool might be able to detect some malware that are not detected by others.
3) Sophisticated obfuscation techniques make detecting malicious codes a difficult task, and generating malware signatures a labor-intensive process.

To tackle the above-mentioned challenges, the computer security researchers have shifted to collaborative approaches [8], [12]–[14], whose main goal is to make different Anti-Virus (AV) tools collaborate and federate their efforts in order to increase the overall detection accuracy, and decrease the required time to feed the viral database with malware signatures. Indeed, collaboration allows, on one hand, to reduce the false positives that are produced when AV tools are not assisted by other sources confirming the alerts about a specific file. On the other hand, the nodes (i.e., machines) whose detection tools are performing poorly will be empowered by correlating the security alerts coming from the well-performing ones in the network. However,

The associate editor coordinating the review of this manuscript and approving it for publication was Luis Javier Garcia Villalba.

to make the collaboration among a large number of detection systems feasible, the following two major issues need to be overcome.

- The collaborators need to agree on the names or identifiers that have to be given to the malware. Indeed, sharing the malware sample is not feasible from a security point of view, in addition to the incurred high network bandwidth consumption, especially in case of large-sized files. Thus, it negatively impacts the scalability of the collaborative mechanism. Unfortunately, AV vendors often report different names for the same malware, which shows the ineffectiveness of the existing naming conventions such as CARO, CME, and MAEC [15]–[17], as discussed in Section II. In addition, they rely on hash functions as the main identification (fingerprinting) technique, which is unable to recognize malware variants. Therefore, there must be coherent mechanisms for integrating information and knowledge, which come from different AV vendors and have different formats, in order to produce a unified identification or naming convention for malware and enable different entities to communicate using a common language.
- The collaborators should agree on which information to share among them and how it is shared. For instance, in addition to the analyzed file identifier, we need to determine other information that are necessary to allow collaboration and achieve a consensual decision on file maliciousness, and how it should be represented. Furthermore, we need to provide an appropriate communication scheme that ensures scalability.

In this paper, we propose MACoMal, a collaborative and fully-distributed decision-making mechanism for enhancing the detection accuracy of anti-malware tools deployed on a network. In order to deal with the malware identification issue, we propose a content-based malware identification approach, i.e., instead of giving a name to the malware by an AV vendor, an identifier is automatically derived from the malware content. In this way, different AV vendors can generate the same identifier for each malware. To this end, we use n-gram opcode-sequence-based signature of the executable file, which is lightweight, harmless, and can also be used to identify malware of the same family. As for the second issue, we present a collaborative decision making scheme based on Multi-Agent Systems (MAS). Our choice is motivated by the fact that MAS can provide the cooperation and collaboration mechanisms that are necessary to reach a consensual decision about the maliciousness of a suspicious file [18]. Thus, we provide a detailed description of the agents' roles and their communication mechanisms, which enable collaborators to reach a consensual decision about malware threats. Indeed, a set of agents are implemented on each host of a network, and each agent has a set of roles, and reacts to a set of events. When the local anti-malware of an agent reports an analyzed executable file as malicious, its identifier (signature) is generated. The signature is then propagated as an alert in the network. When an agent receives an alert, it decides on the maliciousness of the file based on information collected from its neighbors.

The overall architecture of MACoMal consists of the following two main elements:

- *Executable file identification model:* We use the vector space model to represent the opcode sequence of the executable file as an M-dimensional vector. The produced vector can serve to identify the malware and variants of the same malware as well.
- *Collaborative decision-making scheme:* It is composed of two main elements:
  - *Network topology model*: We propose a weighted directed graph model that uses the round-trip time metric to determine which agent interacts with whom.
  - *Agent architecture*: We use the component-level architectural model to describe the components of the agents and the interaction between them.

MACoMal provides the following characteristics:

- It is fully distributed, and does not require any centralized entity. This is a key advantage compared to the existing centralized solutions, such as cloud-based scanning services (e.g., VirusTotal).
- It provides a lightweight file identification and scalable exchange mechanism, as only fixed-size vectors are shared among agents.
- All the components of the architecture can work autonomously without the need for human intervention.
- It is generic and can be adapted at various levels: hosts or vendors, which allows the collaboration between different entities in order to identify the newly collected malware samples.
- It does not require any additional analysis or detection tools. The files are analyzed using the already deployed AV tools at the host machines level. Thus, the architecture can be integrated with local AV tools of heterogeneous nature (from different vendors), allowing them to collaborate in order to reach a consensual decision on a suspicious file.

We formally analyze MACoMal with respect to the following:

- *Network connectivity*: We determine analytically the minimum round-trip time to ensure, with probability p, that the network is connected.
- *Global decision correctness*: We use Linear Temporal Logic (LTL) as a formal tool to prove that all agents will eventually decide on the maliciousness of a file.

In order to evaluate the effectiveness and efficiency of MACoMal, we use a multi-agent simulation tool and a set of real malware samples to measure different metrics, such as the number of nodes reaching global decision over time, the required time and message overhead to reach the global decision, and the precision and recall of the proposed file identification scheme.

The remainder of the paper is organized as follows: Section II presents the related work. Background on Multi-agent systems is given in Section III. In section IV, we present MACoMal's architecture. Section V describes the operations of MACoMal. Formal analysis of MACoMal is provided in Section VI. In section VII, we present the simulation methodology and the results, which are then discussed in section VIII. Finally, Section IX concludes the paper.

II. RELATED WORK

In this section, we present existing research on malware naming and identification, as well as an overview on collaborative architectures for malware detection. This will allow us later to point out the novelty of our work with respect to malware naming and identification, as well as collaborative architectures.

A. MALWARE NAMING

In the past, there have been some attempts to adopt an industry-wide malware naming convention. CARO (Computer Antivirus Research Organization) [15] is one of the naming conventions that dates back to the 1990's, and uses the following format to name malware, as depicted in Figure 1:

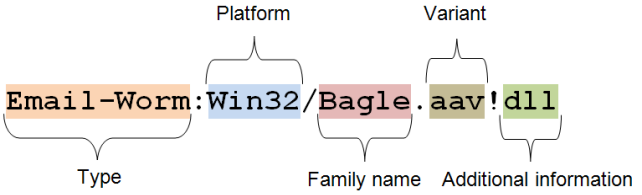


FIGURE 1. CARO malware naming convention [15].

- **Type:** describes what the malware does on the computer. It can be: worm, virus, trojan, backdoor, ransomware, . . ., etc.
- **Platform:** indicates the targeted operating system, such as Windows, Mac OS, and Android.
- **Family:** indicates the group of malware that are based on common characteristics.
- **Variant:** it is a letter that is used sequentially for every different version of a malware family.
- **Additional information:** gives extra details about the malware. For example, ''!lnk'' indicates that the threat component is a shortcut file.

However, CARO was not widely adopted as it was not convincing for anti-malware vendors due to some practical issues to maintain consistency. CME (Common Malware Enumeration) initiative [16] was another effort for assigning identifiers to malware, but it is no longer active. MAEC (Malware Attribute Enumeration and Characterization) [17] is a standard, which extracts attributes from the malware using static and dynamic analysis techniques. The extracted attributes are used to characterize the malware. In this way, malware with the same attributes can be grouped together. However, it is not clear how we can derive malware families from the similar groups. In addition, each of the current dynamic analysis tools generates different attributes, and hence it is unfeasible to update these tools to meet the MAEC standard. So far, the best available tools are services like VirusTotal [19], which cross-references the different names given to malware by Anti-virus vendors.

B. MALWARE FINGERPRINTING

Document fingerprinting consists of mapping data such as documents and files to shorter and unique text strings called fingerprints [20]. There are several methods to generate the fingerprint (or digital signature) of a file. The hash function [21] is one of them. However, it is not resilient against code obfuscation techniques. Indeed, malware variants could have a negative impact on the performance of the anti-virus tools by overloading the analysis table. To deal with the drawback of the traditional hashing algorithms, fuzzy hashing has been proposed to identify two files that are near copies of one another [22]. For instance, SSDeep algorithm [23] has been used by VirusTotal since 2012. Fuzzy hashing techniques still have some limitations due to their fixed-sized fingerprints. This means that if the files are too large or too small, they cannot be meaningfully compared. Also, the existing fuzzy hashing algorithms have been found unsuitable for similarity analysis [24]. In order to verify this claim, we apply the MD5 [25], and SSDEEP hashing algorithms on two variants of the Zbot malware namely: Zbot.aacl and Zbot.aacm. The obtained hash signatures are presented in Table 1.

TABLE 1. Comparison of MD5 signatures of two variants of Zbot malware family.

Malware	Signatures
	MD5 Hash Signature
Zbot.aacl	9eb6fa457757710f3bfcb00c05649533
Zbot.aacm	26ad30c1bb65a193a5f60f7e36c7f004
	SSDEEP Hash Signature
Zbot.aacl	3:N9TDIAwnRzFAjmQvlISASgnkajWLxrPryZNbqR...
Zbot.aacm	3:N9TDIAwnRzFAjmQvlISASgnkajWLxrPryZNbqR...

Results presented in the above table show that the MD5 algorithm generates two completely different hashes, and two with a low similarity (48%) by the SSDeep algorithm. These results justify the need for a more efficient malware files identification scheme.

C. COLLABORATIVE ARCHITECTURES

Some collaborative architectures have been proposed for malware detection. They can be divided into three different categories namely: centralized, hierarchical, and distributed.

1) CENTRALIZED ARCHITECTURES

Oberheide *et al.* [8] introduced a centralized approach for collaborative malware detection based on the Cloud Computing technology. The proposed approach allows analyzing

a suspicious file using several antivirus tools installed on the cloud. Local agents operating on mobile devices act as anti-virus programs by checking the activity of the files on the system. If a suspected file is identified, it will be sent to the network service at the cloud level. The network service is responsible for analyzing files, and checking whether a file is malicious or not. The use of multiple anti-virus engines simultaneously has improved the detection accuracy.

RAVE (Replicated Antivirus for Email Infrastructure) [12] is a centralized system for collaborative malware detection that is deployed at a local network between the Internet and e-mail infrastructure. RAVE is composed of a set of replicas that include two elements: the *Payload* and the *Wormhole*. The first one is used to analyze the files using an antivirus tool. The second one is used to collect the results of the *Payloads* and send them via email to other replicas. The system also has a central entity, which collects the analysis results from the different replicas, and make decision on the maliciousness of the files based on a voting mechanism.

The main drawback of the above centralized architectures is their reliance on a central entity to collect and analyze the suspicious files, and hence they suffer from the single point of failure, which limits their deployment in large-scale networks such as Internet.

2) HIERARCHICAL ARCHITECTURES

Colajanni *et al.* [26] presented a hierarchical (or semi-centralized) collaborative malware detection system based on dynamic analysis, i.e., the suspicious file is run in a controlled environment. In this system, the endpoint machines are considered as sensors. Each sensor acts as a honeypot, which aims at collecting malware samples that attempted an infection. Each group of sensors is connected to a manager, which acts as a ''collection point'' of the malware samples. Managers are themselves connected to a central unit called ''collector'', which analyzes the collected malware samples using sandboxes (i.e., controlled environment). Once an infection is confirmed by the collector, the information is spread to all the subnetworks.

ENDMal [13] is another hierarchical collaborative malware detection system based on dynamic analysis. ENDMaL has a set of lightweight analyzer programs that are implemented at each node of the network, whose role is to analyze the files by extracting their respective sequences of system calls, in order to use them as features for malware detection. ENDMaL is also composed of several monitors, each of which controls a portion of the network and receives the system calls that are extracted by the analyzer programs. Each monitor is composed of an anti-obfuscation mechanism, which is based on a system call alignment method as well as on a probabilistic method for the representation of program behaviors. All monitors collaboratively identify malware families by sharing malware behavior information via a distributed hash table (DHT).

The proposed hierarchical architectures [13], [26] reduce the dependency of the entire network on a central entity. However, the intermediate entities (e.g., monitors) remain the weak points in this type of approach.

3) DISTRIBUTED ARCHITECTURES

Fung *et al.* [14] proposed a distributed system for collaborative malware detection. The purpose is to make different antivirus tools collaborate, which are installed on different machines across a network, in order to improve the detection accuracy. The analysis is carried out by transmitting the suspicious file to the neighboring machines, which in turn, analyze and transmit the file to their neighbors, and so on. The decision regarding the maliciousness of the file is made based on the entire history of results, which are provided by the various anti-virus tools.

The distributed architecture proposed by Fung *et al.* [14] is inadequate for large-scale deployment, since it does not consider a mechanism to identify the files, and merely transfers the entire file from one node to another, which is not feasible in case of a network with limited bandwidth resources. Moreover, it is not convenient from an ethical point of view to transmit a file that is suspected of being malicious.

D. COMPARISON WITH RELATED WORK

Our work differs from the related work in the following points:

- The proposed architecture does not require any additional analysis or detection tools and leverages the already installed malware detection tool. Moreover, these tools can be heterogeneous (i.e., from different vendors).
- We propose a content-based identification scheme, which generates the signature/fingerprint of the program based on its opcode sequences. This signature represents the semantic-level behavior of the program instead of the syntax-level (i.e., hashing), and is more resilient to obfuscation techniques.
- The semantic-based signature serves two purposes: (1) identify the malware, and (2) group variants of the same malware family.
- We propose a lightweight, fully distributed, and collaborative decision making mechanism based on MAS.
- The proposed architecture is generic and can be adopted at various levels (hosts and vendors).

Table 2 provides a qualitative comparison of MACoMal with the previously discussed related work.

III. BACKGROUND ON MULTI-AGENTS SYSTEMS (MAS)

In recent years, multi-agent systems (MAS) have increasingly occupied an important place among the panoply of computer paradigms. MAS have various applications, such as the study of social phenomena, engineering, networks, distributed systems, etc. An agent can be defined as *a physical or virtual entity that can act, perceive its environment (in a partial way) and communicate with others. It is autonomous and has skills to achieve its goals and tendencies* [18].

TABLE 2. Comparison with existing approaches.

Work	Objective	Architecture	File identification	Evaluation Criteria
Oberheide et al. [8]	In-cloud malware scanning and detection	Centralized	None	Scalability
Silva et al. [12]	Malware detection for email infrastructures	Centralized	None	System Latency (in ms)
Colajanni et al. [26]	Malware traffic detection	Hierarchical	None	Malicious connections detection rate
Lu et al. [13]	Malware detection	Hierarchical	None	Detection Rate
Fung et al. [14]	Anti-malware Assistance	Distributed	None	Detection Accuracy
MACoMal	Anti-malware assistance	Distributed	Content-Based	Coverage, Scalability, file identification

Hence, a multi-agent system is composed of intelligent agents that interact to solve problems that may be beyond the capabilities of each individual agent [27].

A. TYPES OF AGENTS

An agent may belong to different categories according to different classification types. Indeed, based on their ability to move within their environment, the agents can be either static or mobile [28]. Moreover, according to the decision-making process, an agent can be either cognitive or reactive [28], [29]:

- **Cognitive agents** have the ability to reason and negotiate as well as exchange knowledge.
- **Reactive Agents**, on the other hand, respond to stimuli coming from their environment.

Some agents may integrate both cognitive and reactive aspects in order to improve action and decision times [30].

B. INTERACTIONS BETWEEN AGENTS

Interaction between agents can take the following forms [18]:

- **Communication:** Agents use two types of communication processes, which are communication by information sharing, and communication by message sending. The first one consists of using a shared workspace, in which each agent drops some information intended to one or several agents. In the second type, the agents use common languages called Agent Communication Languages (ACLs) and communicate by directly sending messages to the recipients. There are three types of messages: questions, answers, and information. There are different types of ACLs as well, however, the FIPA ACL [31], which has been used in our work, is the most used one.
- **Collaboration:** It explains how to distribute work among several agents, whether using centralized or distributed techniques [18].
- **Coordination:** It analyses how the actions of the different agents should be organized in time and space in order to achieve the objectives [18].
- **Cooperation:** It is the most general as well as the most important type of interaction in multi-agent systems. Indeed, cooperation is necessary for the distributed resolution of a problem. This is characterized by the activity of a group of agents converging towards a global goal by achieving their own local goals [32]. There are several models of agents' cooperation, i.e., cooperation by sharing tasks, intermediate results, etc.

IV. MACOMAL'S ARCHITECTURE

In this section, we present MACoMal's architecture, which is composed of the following two elements:

- A collaborative decision-making scheme.
- An executable file identification model.

A. COLLABORATIVE DECISION-MAKING SCHEME

1) NETWORK TOPOLOGY MODEL

We model the network of the collaborative community as a weighted directed graph $G = (V, E, \omega)$, where V represents the set of agents and E represents the set of links. $\omega : E \rightarrow \mathbb{R}$ is a function that maps each link (i, j) to a real value, which is the round-trip time from i to j, denoted by RTT_{ij}, and is defined as the duration it takes for a network request to go from node i to node j and back again to node i. A user/agent can determine the RTT_{ij} by using the known *ping* command.

The above model establishes an *All-to-All* communication scheme, i.e., all agents communicate with each other, which is not scalable with respect to the number of agents. To make the communication scheme scalable, we define RTT_{min} as a threshold that distinguishes between long-delayed and short-delayed links. A link (i, j) is called long-delayed (resp., short-delayed) if $RTT_{ij} > RTT_{min}$ (resp., $RTT_{ij} \leq RTT_{min}$). We also define $G' = (V, E')$, an induced subgraph of G that excludes the long-delayed links. Formally, $E' = E - \{e : \omega(e) > RTT_{min}\}$. A link $(i, j) \in E'$ is established between two neighboring agents i and j if it is short-delayed, i.e., the round-trip time from i to j is upper-bounded by RTT_{min}. The following assumptions are made about the nodes and the network:

1) Communication links are bidirectional: for two neighboring nodes i and j, we have RTT_{ij} and RTT_{ji}, which are both less than or equal to RTT_{min}.
2) Each node has a sufficiently large receive buffer to avoid buffer overflow.
3) The nodes composing the static network are trustworthy, and they are initially approved. Therefore, there is no compromised node in the network.

2) AGENT ARCHITECTURE

In our collaborative model, we have several agents, each of which is deployed on a node (i.e., a machine) in the network. The agents communicate with each other in order to achieve the objectives for which this system is designed, namely to assist the already deployed anti-malware tools to detect the existence of a malicious file within a machine or set

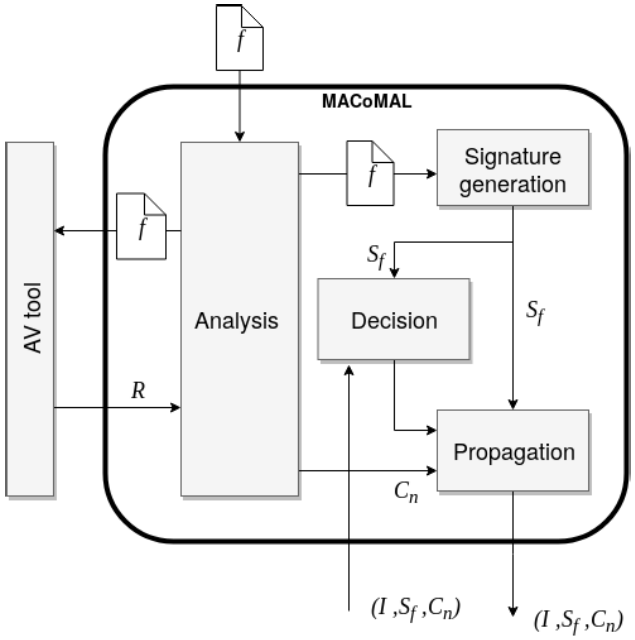


FIGURE 2. MACoMal's architecture (agent-level).

of machines on a network. The overall architecture of the proposed system at the Agent-level is presented in Figure 2.

As shown in Figure 2, each node of the network consists of an agent that plays three roles: (1) files analysis, (2) alerts propagation, and (3) decision on the alert. Each agent calculates its global decision (g), based on three kinds of information received from agent n: (a) agent's id, (b) confidence degree (c_n) of agent n, and the signature of the analyzed file (s_f). All this information together allows to uniquely identify the file being analyzed.

Upon receiving s_f, the agent computes a degree of similarity (ds_n) between the received signature and the stored ones. The agent also stores nbt, which is the number of agents that have triggered an alert on the file (f) or one of its variants. Each node stores the above-mentioned information in a table, called the analysis table, as depicted in Table 3.

TABLE 3. Example of an agent's analysis table.

Signature	c_1	ds_1	...	c_n	ds_n	nbt	g
S_1	0.75	0.78	...	0.93	0.84	7	0.71
S_2	0.63	0.76	...	-	-	1	-

The agents that are deployed at each node are static as well as hybrid. They are considered reactive because they react when a local threat is detected or when an alert is received from a neighboring node. Moreover, they are also cognitive because they store a certain amount of information enabling them to calculate their global decision (g).

The agent responds to two types of events: (1) receiving an alert from one of its neighbors, or (2) receiving locally a file to be analyzed. The first event triggers the *Alert-reception and decision stage*, and the second one triggers the *Analysis stage* that is followed by the *Alert-reception and decision stage*.

B. EXECUTABLE FILE IDENTIFICATION MODEL

Each binary program contains a sequence of opcodes to be executed. Let $\sum = \{op_1, op_1, op_2, \cdots, op_M\}$ be the set of opcodes where $M = |\sum|$ is the number of opcodes.

We formally define the opcode sequence of the program i having K opcodes as: $Seq_i = (op_1, op_2, \cdots, op_K)$, such that op_l is the l^{th} opcode in the sequence.

Let S be the set of M^n distinct n-grams that can be formed from $\sum$. n-grams are all substrings of a larger string with a length of n [33]. For instance the word "MALWARE" can be divided into four 4-grams, which are: "MALW", "ALWA", "LWAR", and WARE. In our case, the string is composed of the sequence of extracted opcodes. Such techniques have been widely investigated for detecting unknown malware or variants of known ones [34], [35].

We use the vector space model to represent the opcode sequence of the executable file as an $|S|$-dimensional vector. To this end, we define $Z_i = a_{i1}, \cdots, a_{i|S|}$ as the profile (or opcode-sequence-based signature) of program i where a_{ij} is the number of occurrence of n-gram opcode j in program i.

We define the function φ that maps each Seq_i to a Z_i vector. Formally:

$$\varphi : \sum^* \rightarrow \mathbb{N}^{|S|}$$
$$Seq_i \rightarrow Z_i$$

We define the similarity measure, denoted by Sim, as a function: $\mathbb{N}^{|S|} \times \mathbb{N}^{|S|} \rightarrow \mathbb{R}$. According to this definition, we define the malware family of Seq_i as follows: $\{Seq_j : Sim(Z_i, Z_j) > \alpha\}$, where α is a predefined value.

V. MACOMAL'S OPERATIONS

In this section, we describe the the following operations that are performed by each agent:

- *Network topology building phase*: It builds the network topology that allows neighboring agents to communicate with each other.
- *Analysis phase*: An agent scans an executable file and checks its maliciousness using a local anti-malware tool.
- *File identification phase*: It generates the n-gram opcode-sequence-based signature of the file (i.e., M-dimensional vector).
- *Agent's confidence degree*: It reflects the performance of the available malware tool determined based on the evaluation proposed by AV-Test.
- *Alert propagation phase*: When an executable file is detected as malicious, its signature is propagated as an alert in the network.
- *Alert-reception and decision phase*: When an agent receives an alert, it decides on the maliciousness of the file based on collected information from its neighbors.

A. NETWORK TOPOLOGY BUILDING PHASE

Initially, each agent i sends a ping packet to each anti-malware agent j in the collaborative community; this allows measuring the round-trip time among agents. If the obtained $RTT_{ij} \leq RTT_{min}$, j is added to the set of i's neighbors, denoted by N_i.

B. ANALYSIS PHASE

This stage consists of scanning the suspicious file using the local anti-malware tool installed on the node (or machine). The analysis result (R) is said to be local and is a Boolean type encoded in binary. In case $R = 1$ (i.e., malicious file is found), the agent propagates this result as an alert to its neighboring nodes, which triggers the *Alert-reception and decision stage* at the level of the receiving nodes. To this end, the agent must first generate the opcode-sequence-based signature of the file (s_f), which allows its identification by other agents of the network.

C. FILE IDENTIFICATION PHASE

Based on the executable file identification model, we extract the operation code (Opcode) sequences from the machine code (X86) of the analyzed file. This is done automatically by a static analysis [36], [37] of the Portable Executable (PE) file [38]. After that, the extracted sequence is represented as n-grams. In Table 4, we compute the 1-gram and 2-grams opcode-sequence-based signatures of Zbot.aacl and Zbot.aacm respectevily, which were already presented in Table 1.

TABLE 4. n-grams Opcode-sequence-based signatures of Zbot.aacl et Zbot.aacm malware.

Malware	Opcode-sequence-based signature
Zbot.aacl (1-gram)	`add(19),sub(3),mov(2),...`
Zbot.aacl (2-grams)	`add add(12),add int3(2),...`
Zbot.aacm (1-gram)	`add(23),xor(4),pop(3),...`
Zbot.aacm (2-grams)	`add add(10),add int3(2),...`

The signature is composed of the names of the operands as well as their number of occurrences in the code. For example, `add (19)` means that the `add` operand appeared 19 times in the Zbot.aacl malware code, and `add add(12)` means that the bi-gram `add add` appeared 12 times in the sequence of the same malware.

D. AGENT'S CONFIDENCE DEGREE (C_N)

The confidence degree of a node (c_n) reflects the performance (accuracy) of the available anti-malware tool, which is determined periodically based on the evaluation proposed by AV-Test.[1] This evaluation is carried out according to three different criteria which are: protection, performance, and usability. In our case, and in order to determine the confidence degree of a node (c_n), we consider protection as the main evaluation criterion, since it reflects the detection accuracy of an anti-malware tool, and its value ranges between 0.5 and 1. This range has been chosen according to the values provided by AV-Test. Indeed, no AV tool has a protection score less than 3 on a scale of 6 (3/6).

E. ALERT PROPAGATION PHASE

Once a file is detected by the local anti-malware tool as malicious, the confidence degree (c_n) is computed, and the file's signature (s_f) is generated. The agent triggers the alert-reception and decision stage, which is responsible for the decision-making and the propagation of the alert to the neighboring nodes. The alert is represented by a triplet (agent id, s_f, c_n). The analysis stage is summarized in Algorithm 1.

Algorithm 1 *Local_Analysis*(f)

```
R ← analyze(f)
if (R = 1) then
    S_f ← signature(f)
    block(f)
    Recieve_Alert_Decision(id, S_f, c_n)
end if
```

In case the file is analyzed locally by agent id_n, the *Recieve_Alert_Decision*() is invoked, and takes as parameters: the id of the agent, and the confidence degree (id_n,s_f, c_n). However, in case the alert is received from another node, the same method will be invoked using the received id and confidence degree (id_t,s_f,c_t), which are related to the node that triggered the alert.

F. ALERT-RECEPTION AND DECISION PHASE

When an agent receives an alert that is triggered by a locally analyzed file, or an alert that comes from a neighboring node, the agent will look up the signature of the file in its analysis table. It computes the pairwise comparison, i.e., degree of similarity (ds_n), which varies between 0.0 and 1.0. The cosine similarity is calculated using Equation 1.

$$ds(S_f, S_{f'}) = \cos(v, u) = \frac{v.u}{\|v\|\|u\|} = \frac{\sum_{i=1}^{n} v_i u_i}{\sqrt{\sum_{i=1}^{n} (v_i)^2}\sqrt{\sum_{i=1}^{n} (u_i)^2}} \quad (1)$$

After computing the degree of similarity (ds_n), and if there is similarity between the received signature and one of the signatures that are stored in the analysis table ($ds_n > \alpha$), then the agent counts the number (nbt) of nodes that have triggered an alert about that file or one of its variants. If (nbt) is greater than a certain threshold δ, then the agent computes a global decision (g), using Equation 2.

$$g = \frac{\sum c_n \times ds_n}{\sum c_n} \quad (2)$$

Then, g is compared to a threshold β. If g is greater than β, then the agent concludes that the file is malicious, and sets the value of g to 1 in order to influence the decisions of its neighboring nodes. The agent, then, communicates its decision using the *Send_Alert*() method (see Algorithm 2) along with the triplet (agent id, s_f, 1). The *Send_Alert*() method invokes the *Recieve_Alert_Decision*() method of the neighboring agents (N_i) and uses as parameters the same triplet (agent id, s_f, 1).

In case there is no similarity between the signature of the analyzed file and the signatures stored at the analysis table ($ds \leq \alpha$), the agent adds the received signature to the table along with the received (c_n) and propagates the

[1] https://www.av-test.org/en/antivirus/home-windows/

Algorithm 2 *Send_Alert*(id, s_f, c_n, N_i)

1: $@N_i$.*Recieve_Alert_Decision*(id, S_f, c_n)

Algorithm 3 *Recieve_Alert_Decision*(id_n, s_f, c_n)

```
for i = 1, SizeOF(Analyses_Table) do
    if match(Analysis_Table[i].Signature,s_f,&ds_n) then
        match ← true
        Analysis_Table[i].nbt++
        Analysis_Table[i].c_n ← c_n
        Analysis_Table[i].ds_n ← ds_n
        if (Analysis_Table[i].nbt > δ) then
            Calculate(g)
            if (g > β) then
                Analysis_Table[i].g ← 1
                remove(f)
                for all agents(N) neighboring(A) do
                    Send_Alert(id_A, S_f, 1, N_A)
                end for
                break()
            else
                Analysis_Table[i].g ← 0
                unblock(f)
                break()
            end if
        end if
    end if
end for
if match = false then
    Analysis_Table[i+1].Signature ← S_f
    Analysis_Table[i+1].c_n ← c_n
    Analysis_Table[i+1].nbt++
    for all agents (N) neighboring (A) do
        Send_Alert(id_n, S_f, c_n, N_A)
    end for
end if
```

alert to its neighbors using the *Send_alert*() method. In this case, the latter method takes as parameters the same received triplet composed of the id of the node that triggered the alert, the received (c_n), and the file's signature (s_f). The alert-reception and decision stage is presented by a method named *Recieve_Alert_Decision*() in Algorithm 3.

VI. FORMAL ANALYSIS

A. GRAPH CONNECTIVITY ANALYSIS

In this section, we investigate the graph connectivity problem, i.e., we want to determine analytically the value of RTT_{min} to ensure that the network is connected.

Based on the defined link model, two nodes are neighbors if $RTT_{ij} \leq RTT_{min}$ and $RTT_{ji} \leq RTT_{min}$. The Round trip time (RTT) can be developed as follows:

$$\begin{aligned} RTT &= 2\,PDT + PT \\ &= 2(PTT + PD) + PT \\ &= 2(PTT + \frac{D}{PS}) + PT \end{aligned}$$

such that: PDT, PT, PTT, PD, D, and PS denote the *packet delivery time*, *processing time*, *packet transmission time*, *propagation delay*, *distance*, and *propagation speed* respectively. For the sake of simplicity, we consider that $PT \ll PDT$. We assume that PS ranges between 2×10^8 meters/sec for copper wires and 3×10^8 meters/sec for wireless communication. By considering an ultra-conservative estimate of RTT_{ij}, we set PS to 2×10^8, and PTT to $\max(PTT)$, which is the upper-bound of PTT.

As PS and PTT are constant, RTT_{ij} can be mapped to the equivalent distance D_{ij} between nodes i and j.

Therefore, our problem is converted to the following:

What is the minimum distance D_{min} required to ensure that the network is connected? This problem is solved using *the nearest neighbor method* [39].

In the network, a set of n nodes are randomly distributed in a region of area A. Node density $\lambda = \frac{n}{A}$. For a homogeneous Poisson point process in $\mathbb{R}^2$ with constant λ, the probability density function of the nearest neighbor distance x is written as:

$$f(x) = 2\pi\lambda D e^{-\lambda\pi x^2} \tag{3}$$

The probability that the distance between a randomly chosen node to its nearest neighboring node is less than or equal to D, is given by

$$P(x \leq D) = \int_{x=0}^{D} f(x)dx = 1 - e^{-\lambda\pi D^2} \tag{4}$$

The degree of node u, denoted as $\Delta(u)$, is the number of its neighbors. A network is connected if all the nodes are not isolated, i.e., $\forall u \in G : \Delta(u) > 0$. If we consider D_{min} as the minimum distance required to ensure the network connectivity, we have:

$$P(\Delta(u) > 0) = P(x \leq D_{min}) \tag{5}$$

The probability that a node is isolated, i.e., has no neighbor, is

$$\begin{aligned} P(\Delta(u) = 0) &= P(x > D_{min}) \\ &= 1 - P(x \leq D_{min}) \\ &= e^{-\lambda\pi D_{min}^2} \end{aligned} \tag{6}$$

The minimum node degree of G is denoted as

$$\Delta_{min} = \min_{\forall u \in G}\{\Delta(u)\} \tag{7}$$

The probability that no node is isolated in G is equivalent to the event that $\Delta_{min} > 0$

$$P(\Delta_{min} > 0) = (1 - e^{-\lambda\pi D_{min}^2})^n \tag{8}$$

From Equation 8, we derive D_{min}, the minimum distance to ensure with probability $P(\Delta_{min} > 0)$ that the network is connected

$$D_{min} = \sqrt{\frac{-\ln(1 - P(\Delta_{min} > 0)^{\frac{1}{n}})}{\lambda\pi}} \tag{9}$$

From D_{min}, we derive RTT_{min} as follows:

$$RTT_{min} = 2(\max(PTT) + \frac{D_{min}}{2 \times 10^8}) \tag{10}$$

B. GLOBAL DECISION CORRECTNESS

In this section, we use the Linear Temporal Logic (LTL) [40] as a formal tool to verify the correctness of MACoMal. A temporal formula consists of predicates, and temporal operators like $\Diamond$ ('eventually'), and $\Box$ ('at every moment in the future'), quantification operators ($\forall, \exists$), and boolean operators ($\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$). In the proof, we are going to use propositions:

- *Invariance:* $\Box\varphi$ means that φ is true at every moment in the future.
- *Guarantee:* $\Diamond\varphi$ means that φ will eventually be true at some moment in the future.
- *Stability:* $\Diamond\Box\varphi$ means that φ will eventually be true forever.
- *Response:* $\varphi \Rightarrow \Diamond\psi$, means that when φ is true, ψ will be true at some moment in the future.

The various states assigned to an agent with respect to a given alert are depicted by a Finite State Machine (FSM), as shown in Figure 3. The transitions between states are labeled with triggering conditions, which need to hold true. The states of the FSM are described below:

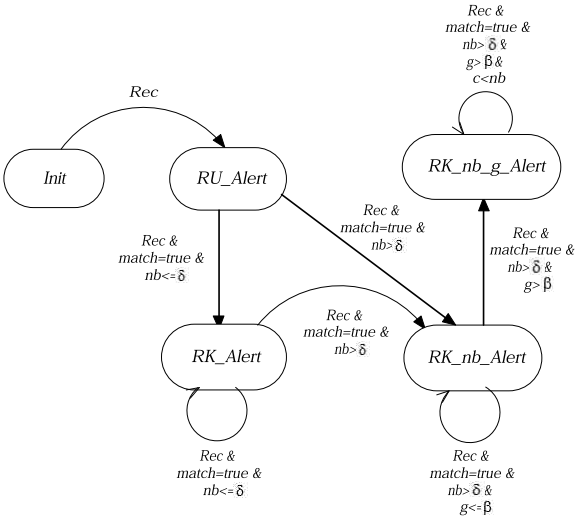


FIGURE 3. Finite state machine of MACoMal.

- *Init*: In this state, the agent has no knowledge of the alert.
- *RU_Alert*: The agent enters this state when it receives an unknown alert. This state corresponds to lines 19-24 of Algorithm 3.
- *RK_Alert*: The agent enters this state when it receives a known alert less than $(\delta + 1)$ times. This state corresponds to lines 2-6 of Algorithm 3.
- *RK_nb_Alert*: The agent enters this state when it receives a known alert more than δ times. This state corresponds to lines 7-8 and lines 15-18 of Algorithm 3.
- *RK_nb_g_Alert*: The agent enters this state when it receives a known alert more than δ times and $g > \beta$. This state corresponds to lines 9-14 of Algorithm 3.

Before verifying the security properties, we formally define other predicates and variables employed by our proof.

- Predicates I_n, RU_n, RK_n,RK_nb_n, and $RN\_nb\_g_n$ hold true if agent n is in the *Init*, *RU_Alert*, *RK_Alert*, *RK_nb_Alert*, and *RK_nb_g_Alert* respectively.
- *Send*: a predicate that holds true at the instant when the agent sends an alert.
- *Recv*: a predicate that holds true at the instant when the agent receives an alert.
- *Nb_Alert*: The number of agents that trigger the same alert i.

Assuming that the network is connected, and each agent n starts execution in a state satisfying predicate I_n, we prove the following theorem:

Theorem 1 (Global Decision Theorem): Whenever δ agents trigger the same alert, all agents will eventually conclude that the alert is a malware.

Formally:

$$P \equiv \Box(\forall n \in V : I_n \wedge (Nb\_Alert \geq \delta) \Rightarrow \Diamond RN\_nb\_g_n) \tag{11}$$

Lemma 1: When an alert is generated by a node s, each node $n \in V$ will eventually converge to a state satisfying the predicate RU_n.

Formally,

$$Q \equiv \forall n \in V : I_n \wedge Recv \Rightarrow \Diamond(RU_n \wedge Send) \tag{12}$$

Proof:

Definition 1: The set of nodes that are within $l-$ hops away from s is denoted by H_s^l.

Let L_s be the longest path in terms of number of hops between s and any node in the network. Hence. $H_s^{L_s} = V$.

Let r be the number of time units after node s has generated an alert. We will show by induction on the number of rounds $0 \leq r < L_s$ that:

$$Q_r \equiv \forall n \in H_s^r : (I_n \wedge Recv) \Rightarrow \Diamond(RU_n \wedge Send) \tag{13}$$

Basic case: $r = 0$. Using the definition of H_s^r, we have $H_s^0 = \{s\}$. Agent s locally receives the alert, executes the code related to *RU_Alert* state, and propagates the alert to its neighbors. Hence, Q_0 holds true.

Inductive hypothesis: Let us assume that Q_r holds true for all $r < L_s$. We will now prove that:

$$Q_{r+1} \equiv \forall n \in H_s^{r+1} : (I_n \wedge Recv) \Rightarrow \Diamond(RU_n \wedge Send) \tag{14}$$

$$Q_r \Rightarrow \forall n \in H_s^r : Recv \wedge RU_n \wedge Send \tag{15}$$

Statement 15 means that when node n at r-hop away from s receives the alert, it enters *RU_Alert* state, and propagates the alert to its neighbors including $(r + 1)$-hop away from s. Hence, From 15, it follows that:

$$\forall n \in H_s^{r+1} : (I_n \wedge Recv) \Rightarrow \Diamond(RU_n \wedge Send) \tag{16}$$

When $r = (L_s - 1)$, we have $H_s^{L_s} = V$, and hence Q holds trues, and Lemma 1 is proved. □

Lemma 2: When at least δ agents trigger the same alert, each node $n \in V$ will eventually converge to a state satisfying the predicate RK_nb_Alert.

Proof: By lemma 1, all the triggering agents of the same alert will reach all the agents $n \in V$. We will now consider two cases:

- $\delta = 1$, each agent will eventually enter the *RK_nb_Alert* state.
- $\delta > 1$, each agent will eventually enter the *RK_Alert* state. As *Nb_Alert* is an increasing variable, when $Nb\_Alert \geq \delta$, each node $n \in V$ will eventually enter the *RK_nb_Alert* state.

Formally, we have:

$$\forall n \in V : RU_n \wedge (Nb\_Alert \geq \delta) \Rightarrow \Diamond RN\_nb_n$$

and hence Lemma 2 is proved. □

Lemma 3: When node n is in the RK_nb_Alert state and its global decision value g is higher than β, it will eventually converge to a state satisfying the predicate $RN\_nb\_g_n$.

Proof: Upon receipt of an alert in the *RK_nb_Alert* state, we consider two cases:

- $g < \beta$, node n remains in *RK_nb_Alert* state.
- $g \geq \beta$, node n transits to *RK_nb_g_Alert* state, sets g to 1 and sends an alert with $c_n = 1$ to its neighbors. The receiving nodes will recompute g with a new value (i.e, $c_n = 1$), and hence they will get a higher value of g.

We can notice that g is not a decreasing function. When a node n receives an alert, g can take two values; remains unchanged or is set to 1.

- g is set to 1.
- It remains unchanged. If we consider the case of highly similar variants and trustworthy antivirus vendors, then all nodes will have $g \geq \beta$, and hence g is set to 1.

As g is not a decreasing function, once it is set to 1, it will remain unchanged forever. Formally, $\Diamond\Box(g = 1)$ holds true, which also means that once an agent enters *RK_nb_g_Alert* state, it will stay there forever. Formally, we have

$$\forall n \in V : \Diamond\Box RN\_nb\_g_n \tag{17}$$

Therefore, the previous three lemmas prove Theorem 1. □

VII. SIMULATION

In this section, we present the tools, parameters, and the methodology that are used to simulate the proposed collaborative mechanism for anti-malware assistance. We also assess its effectiveness and efficiency using real malware samples as well as simulation-based experiments. We measure the performance of the collaborative system in terms of the following four metrics:

- The precision and recall of the proposed opcode-based file identification scheme.
- The number of nodes that succeed in reaching a global decision about a suspicious file (deciding nodes).
- The required time to reach a global decision.
- The required number of messages to reach a global decision.

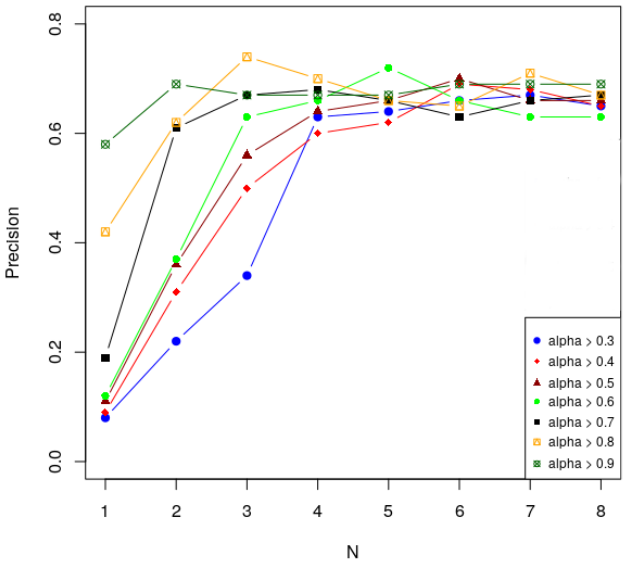


FIGURE 4. Obtained malware identification results (precision) using n-gram opcode-sequences signatures and cosine similarity metric.

A. EFFECTIVENESS OF THE PROPOSED FILE IDENTIFICATION MECHANISM

In order to show the ability of the proposed n-gram opcode-sequence-based signature to identify malware variants, we compute the degree of similarity (*ds*), using Cosine similarity metric [41], between the n-gram opcode-sequence-based signatures of 395 malware divided into 26 families[2] and that for different values of n, and different similarity thresholds (α). We used both precision (P) and recall (R) as the main metrics to evaluate the correctness of the identification scheme. Precision (see Equation 18) allows to evaluate how well unknown malware are assigned to the different families. Recall (see Equation 19) determines how well malware from the same family are grouped together.

$$P = \frac{No.\ of\ malware\ correctly\ assigned\ to\ family\ i}{Total\ no.\ of\ malware\ assigned\ to\ family\ i} \tag{18}$$

$$R = \frac{No.\ of\ malware\ correctly\ assigned\ to\ family\ i}{Total\ no.\ of\ malware\ belonging\ to\ family\ i} \tag{19}$$

We remind that our objective is to reduce the size of the analysis table by assigning newly analyzed malware samples to existing malware families. The experimental results are presented in Figure 4, and Figure 5.

From the above results, we can observe that the highest accuracy (74%) is obtained using 3-grams ($n = 3$) opcode signature with a similarity threshold (α) greater than 0.8, and the recall value is equal to 47%. We believe that the obtained results are satisfactory and the proposed n-gram opcode-sequence-based identification can provide more efficient way to uniquely identify malware samples and their variants compared to hashing techniques, which generate a signature for each malware sample.

[2]Malware samples used in this experiment were obtained from: https://github.com/opsxcq/mirror-vxheaven.org

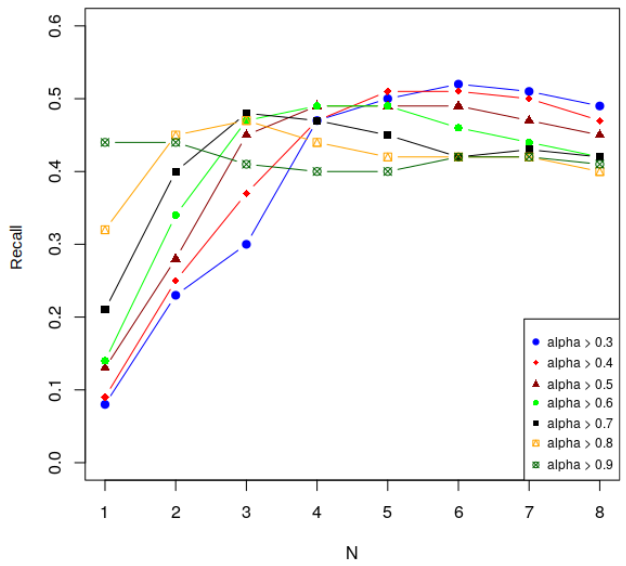


FIGURE 5. Obtained malware identification results (recall) using n-gram opcode-sequences signatures and cosine similarity metric.

B. EFFECTIVENESS OF THE DECISION MAKING PROCESS

1) SIMULATION TOOL

In order to evaluate the performance of the proposed collaborative decision-making scheme, we conduct a set of simulation-based experiments. We use, for that purpose, a well-known simulation environment namely NetLogo [42]. For the visualization part, our simulation is inspired by the Netlogo model "Virus on Network" [43]. In this model, a set of nodes is randomly deployed in the network. A node is randomly chosen and connected to the nearest node in terms of Euclidean distance. This operation is repeated until the average node degree, which is an input parameter of the model, is reached. In addition, to simulate the collaboration model, we follow the FIPA ACL (Agent Communication Language Specification) [31]. For that purpose, we use a third party Netlogo library that is provided by [44]. In Figure 6, we present the simulation program control window.

Note that the simulations are conducted on the following hardware and software configurations:

- **Operating system:** 18.04.3 LTS 64 bits
- **Netlogo:** v. 5.3.1
- **Processor:** Intel Core i5-4300U CPU 1.90GHz × 4
- **Memory:** 8 Gb

2) SIMULATION PROCESS

By using the simulation tool, the simulation process consists of running the collaborative system under different execution scenarios. For every scenario, we use a different set of parameters. Table 5 provides a description as well as the possible values of the different parameters, which are used by the collaborative system during the execution scenarios. The parameter *nbt* represents the number of nodes that triggered an alert regarding a suspicious file. The value of *nbt* needs to exceed a certain threshold (δ), which represents the minimal number of triggers required to compute the global decision (g). We set this value to $\frac{1}{2}nbt$, which means that we need to get alerts coming from at least more than half of the triggers. This is required to make sure that the agent collected enough information coming from different sources about a file (or one of its variants) to compute the global decision (g). Regarding the threshold β, the choice of the range 0.6-1.0 is made to avoid high false alarms (when the threshold is too low) and low detection rate (when the threshold is too high).

TABLE 5. Collaborative system parameters.

Param.	Description	Value
Nbn	The number of nodes forming the network	2 - ...
Nbt	The number of nodes that triggered an alert	1 - Nbn
δ	Minimum number of triggers required to calculate global decision	$\frac{1}{2} \times Nbt$
β	Threshold to decide if the file is malicious or not	0.6 - 1.0

For every simulation scenario, we compute the required time for all nodes to reach a global decision, as well as the average number of messages exchanged per node and per second. The main aim is to show that our proposed collaborative system is able to make a decision about a malware in a reasonable time. Moreover, we generate a plot for each simulation scenario representing the number of nodes that reached a global decision over time (in seconds).

3) EXAMPLE OF SIMULATION SCENARIO

In this illustrative example, we assume that our network is composed of four nodes 1, 2, 3, 4. The parameters, which are used in the simulation scenario, are presented in Table 6

In this scenario, and as shown in Figure 7, agent 1 triggers the analysis by generating an alert message and propagates it to its neighboring agents (Figure 7-a). The message contains information about the sender, a list of receivers (in our case nodes 2, 3, and 4), the id of the initiator agent, the signature of the analyzed file (referred to as *Sf*), and finally the confidence level of the initiator (in case of node 1, c = 0.9). The generated message is presented below.

`[inform sender: 1 receiver 2 receiver 3 receiver 4 content [1,` s_f`, 0.9]]`

For the sake of clarity, in the scenario (Figure 7), we only display the content part of the message, which contains the ID of the initiator agent, the signature of the analyzed file (referred to as s_f), and the confidence level of the initiator agent.

Once the neighboring nodes receive the alert message, they will automatically store it into their respective incoming message queue. Every node that receives the alert message will extract different information from it, such as the sender's ID, the list of receivers, C_n, and the file's signature (S_f). The agent then triggers the *Recieve_Alert_Decision*() method presented in Algorithm 3.

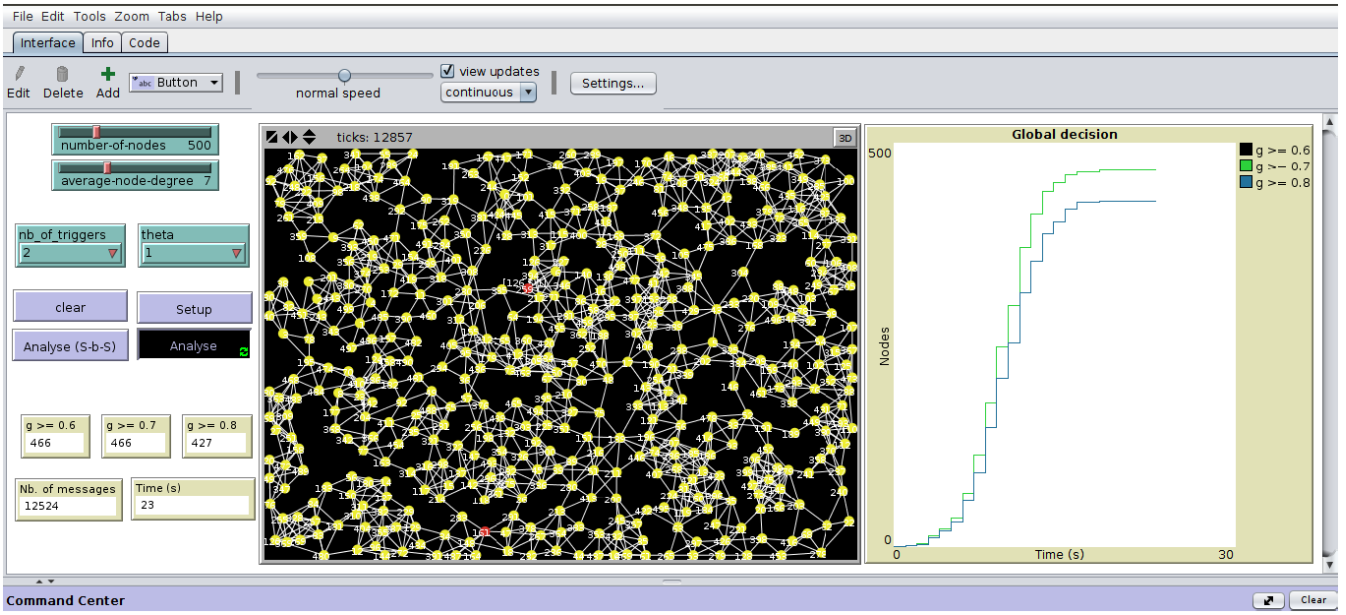


FIGURE 6. Overview of the simulation program control window.

TABLE 6. Parameter values of the example scenario.

Trig.	α	δ	β	c_n	ds_1	ds_2	ds_3	ds_4
1	0.7	1	0.7	0.9	-	0.8	0.9	0.8
2	0.7	1	0.7	0.7	0.9	-	0.9	0.8

Based on our scenario, agents 2, 3, and 4 will store the content of the received message in the analysis table (since the analysis table is empty, the signature is directly added and and its similarity degree is set to 1). At $t = 2$, and $t = 3$ (Figure 7-b), node 2 also triggers an alert and propagates it to its neighbors (agent 1), which in turn propagates it to its neighbors (agents 3 and 4). Once they receive the agent's 2 alert, they will proceed to the computation of their corresponding similarity degree (ds) between the previously received signature (agent 1 alert) and the current signature. In our scenario, the two signatures will match ($ds \geq 0.8$), and therefore, the record in the analysis table corresponding to that signature will be updated. Moreover, both nodes will calculate their global decision (g). In the case of agent 3, $g = (0,9 \times 0,9) + (0,7 \times 0,8)/(0,7 + 0,9) = 0.9$, which is greater than 0.7. Therefore, it will propagate its alert (Figure 7-c), which will be followed by the propagation of other alerts: agent 4 with $g = 0.8$, then agent 2 with $g = 0.8$, and finally, agent 1 with $g = 0.9$. At t = 9 (Figure 7-f), all the agents calculated their global decision and had $g > 0.7$, which means that all the agents are able to detect the malware and its similar variants.

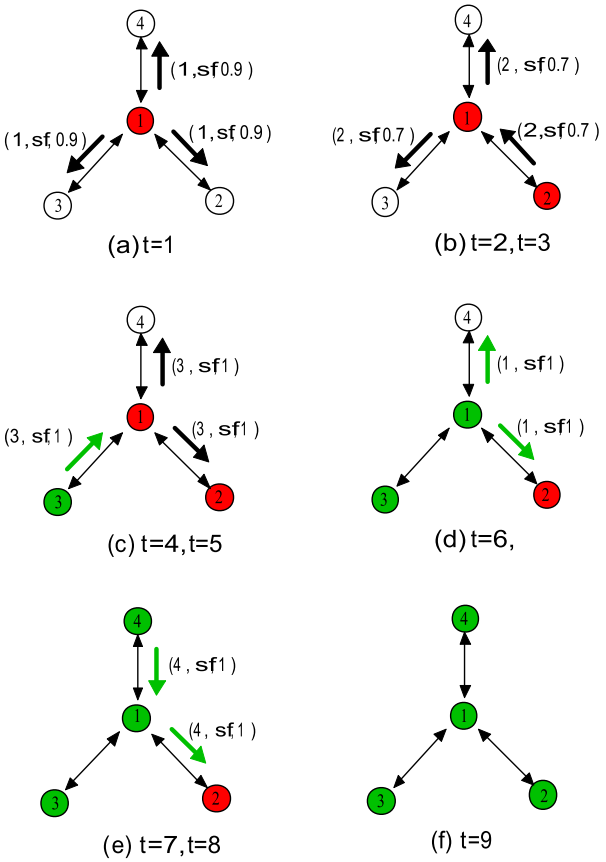


FIGURE 7. Example of collaboration scenario composed of 4 nodes.

4) SIMULATION RESULTS

The obtained simulation results for the different simulation scenarios are presented in Table 7, Figure 8, Figure 9, Figure 10, and Figure 11.

By analyzing the results presented in Figures 8, 9, 10, and 11, we can see that in all scenarios, all the nodes can reach a global decision in limited time period, which ranges from

TABLE 7. Experimental results using the simulation tool.

Nb. of nodes	Nb. of triggers	δ	β	Nb. of messages	Time (s)	Avg. Msg/node/sec
500	2 (≈0.5%)	1	0.6	15182	26	≈ 1.16
500	2 (≈0.5%)	1	0.7	17438	27	≈ 1.29
500	2 (≈0.5%)	1	0.8	21583	73	≈ 0.59
500	5 (1%)	2	0.6	22662	41	≈ 1.05
500	5 (1%)	2	0.7	24904	42	≈ 1.18
500	5 (1%)	2	0.8	27355	57	≈ 0,95
1000	5 (0.5%)	2	0.6	47201	234	≈ 0.20
1000	5 (0.5%)	2	0.7	51817	234	≈ 0.22
1000	5 (0.5%)	2	0.8	57665	374	≈ 0.15
1000	10 (1%)	5	0.6	91944	425	≈ 0.21
1000	10 (1%)	5	0.7	96552	425	≈ 0.22
1000	10 (1%)	5	0.8	98311	425	≈ 0.23

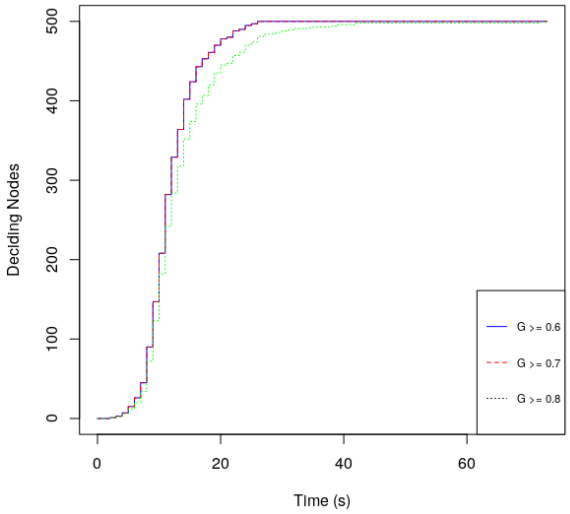


FIGURE 8. Obtained results of the first simulation scenario (nbn = 500, nbt = 2 (0.5%), $\delta = 1$).

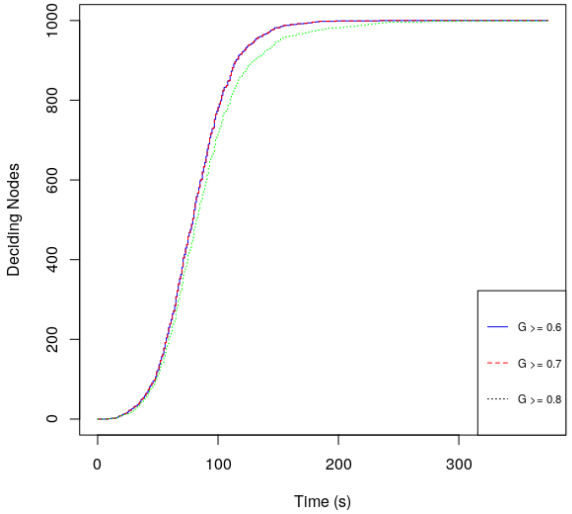


FIGURE 10. Obtained results of the third simulation scenario (nbn = 1000, nbt = 5 (0.5%), $\delta = 2$).

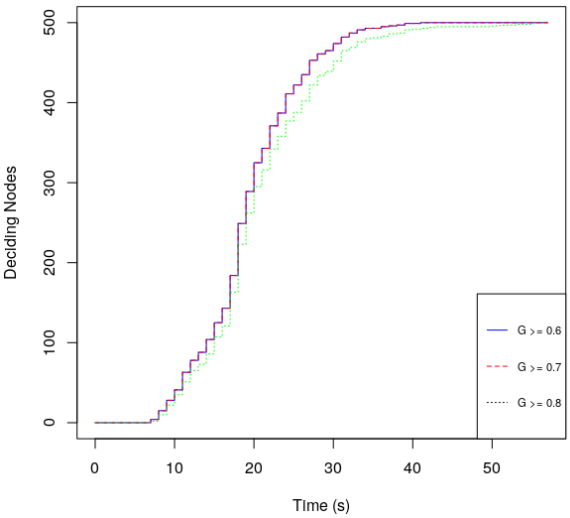


FIGURE 9. Obtained results of the second simulation scenario (nbn = 500, nbt = 5 (1%), $\delta = 2$).

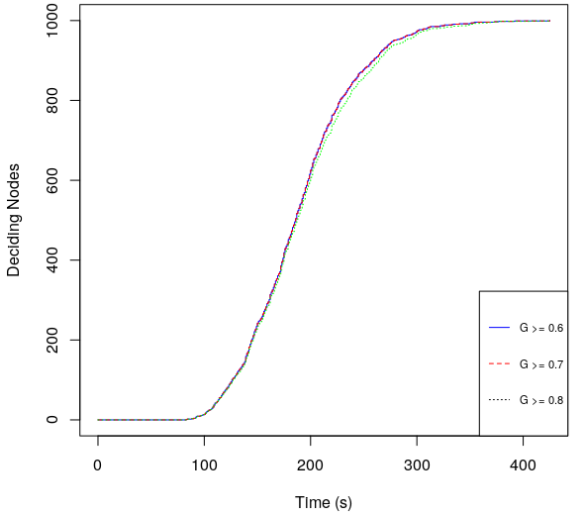


FIGURE 11. Obtained results of the fourth simulation scenario (nbn = 1000, nbt = 10 (1%), $\delta = 5$).

26s to 425s. These results support what has been previously stated, and formally proved. Moreover, we notice that our system converges more quickly when we use small number of triggering nodes (*nbt* equals to 0.5% of the total nodes number, see Figure 8, Figure 10).

From Table 7, we can notice that in case of the first simulation scenario (nbn = 500, nbt = 2 (0.5%), $\delta = 1$) 15182 messages are generated. If the analysis is made based on the number of nodes, each node generates on average ≈ 30 messages. If we divide this number by the time required

for g to reach the first threshold ($\beta \geq 0.6$), we obtain $\approx$ 1 message generated per node per second. The latter value is almost the same for the second threshold ($\beta \geq 0.7$), and it goes down by half in case of the third threshold ($\beta \geq 0.8$). Moreover, it also decreases when the number of triggering nodes is increasing (nbt = 5). In case of the fourth scenario (nbn = 1000, nbt = 10 (1%), $\delta = 5$), the average number of messages /node /second ranges between 0.15 and ≈ 0.23, which can be 80% less compared with the first scenario.

VIII. DISCUSSION

As mentioned previously, the consensual decision within the network is achieved by calculating and spreading the global decision (g). The latter allows to reduce the false positives that are produced in the case of isolated AV, which are not assisted by others that can confirm the malware alert. Moreover, the detection rate is also increased, since all the nodes are able to decide about the presence of a malicious file. In this case, the agents whose detection tools are poorly performing will be empowered by correlating the security alerts coming from the well-performing agents of the network. It should be noted here that the collaboration within the community of agents is indirect. The influence of agents on others is done by spreading the global decision (g), which emerges within a subset of agents that can influence others. As a result, and contrary to other approaches where collaboration requires direct negotiation between agents, in our case, it involves a sort of vote, weighted by the degree of performance of the detection tools.

The use of an adequate file identification scheme is important to our collaborative system. Indeed, it is necessary that the collaborative agents within the system agree on the identity of the code being analyzed. If this characteristic does not exist, we will be forced to analyze any code within the network, which incurs a heavy bandwidth load, overloading the analysis table, and harming the users. From the experimental results, the proposed n-gram opcode-sequence-based signature model has shown good performance compared to the hash function ones.

The obtained simulation results show that after a short period of time (few minutes at most), a network which is composed of a thousand of nodes, will be completely immunized against a malicious threat that has been detected by a relatively small number of nodes (0.5% - 1% of total number of nodes). These results are highly satisfactory compared to the time required by the conventional signature-based malware detection schemes. By analyzing the number of messages that are generated and exchanged among nodes, we can conclude that our collaborative system is scalable and does not incur any network traffic overhead while increasing the number of nodes. Indeed, the number of generated messages per node and per second decreases by 80% when we double the number of nodes in the network.

IX. CONCLUSION

In this paper, we have proposed a methodology, named MACoMal, to model, analyze, and simulate a multi-agent based collaborative mechanism that assists anti-malware tools. It relies on a collaborative model that considers three elements: network topology model, agent architecture, and executable file identification model. MACoMal defines the agents' roles and their communication mechanisms, which allow to build the network topology and their collaboration in order to reach a consensual decision about the maliciousness of a suspicious file. It also proposes an opcode-sequence-based signature model that allows identifying the malware and its variants of the same family.

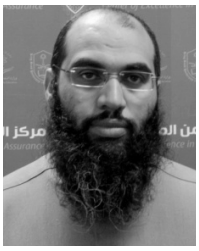
MACoMal is formally analyzed with respect to network connectivity and global decision correctness. The simulation results show the effectiveness and efficiency of MACoMal in terms of precision, recall, number of nodes reaching global decision over time, and the required time and message overhead to reach the global decision. The results demonstrate that a network composed of a thousand of agents can be immunized within a time that ranges from few seconds to few minutes, which is far better than the conventional signature-based malware detection schemes.

REFERENCES

[1] Z. Salehi, A. Sami, and M. Ghiasi, "Using feature generation from API calls for malware detection," *Comput. Fraud Secur.*, vol. 2014, no. 9, pp. 9–18, Sep. 2014.
[2] J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity," *J. Comput. Syst. Sci.*, vol. 80, no. 5, pp. 973–993, Aug. 2014.
[3] J. M. Kizza, *Guide to Computer Network Security*. London, U.K.: Springer-Verlag, 2015.
[4] C. Easttom, "The role of weaponized malware in cyber conflict and espionage," in *Proc. 13th Int. Conf. Cyber Warfare Secur. (ICCWS)*, 2018, p. 191.
[5] H. Kettani and P. Wainwright, "On the top threats to cyber systems," in *Proc. IEEE 2nd Int. Conf. Inf. Comput. Technol. (ICICT)*, Mar. 2019, pp. 175–179.
[6] M. Belaoued, A. Boukellal, M. A. Koalal, A. Derhab, S. Mazouzi, and F. A. Khan, "Combined dynamic multi-feature and rule-based behavior for accurate malware detection," *Int. J. Distrib. Sensor Netw.*, vol. 15, no. 11, Nov. 2019, Art. no. 155014771988990.
[7] *Internet Security Threat Report*, Symantec, Mountain View, CA, USA, Feb. 2019, vol. 24.
[8] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in *Proc. USENIX Secur. Symp.*, 2008, pp. 91–106.
[9] B. Potter and G. Day, "The effectiveness of anti-malware tools," *Comput. Fraud Secur.*, vol. 2009, no. 3, pp. 12–13, Mar. 2009.
[10] D. Carlin, A. Cowan, P. O'kane, and S. Sezer, "The effects of traditional anti-virus labels on malware detection using dynamic runtime opcodes," *IEEE Access*, vol. 5, pp. 17742–17752, 2017.
[11] A. Acar, L. Lu, A. S. Uluagac, and E. Kirda, "An analysis of malware trends in enterprise networks," in *Proc. Int. Conf. Inf. Secur.* Cham, Switzerland: Springer, 2019, pp. 360–380.
[12] C. Silva, P. Sousa, and P. Veríssimo, "Rave: Replicated antivirus engine," in *Proc. Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2010, pp. 170–175.
[13] H. Lu, X. Wang, B. Zhao, F. Wang, and J. Su, "ENDMal: An anti-obfuscation and collaborative malware detection system using syscall sequences," *Math. Comput. Model.*, vol. 58, nos. 5–6, pp. 1140–1154, Sep. 2013.
[14] C. J. Fung, D. Y. Lam, and R. Boutaba, "Revmatch: An efficient and robust decision model for collaborative malware detection," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*. IEEE, May 2014, pp. 1–9.
[15] L. Zeltser. (2011). *How Security Companies Assign Names to Malware Specimens*. [Online]. Available: https://zeltser.com/malware-naming-approaches/
[16] (2006). *Common Malware Enumeration (CME)*. [Online]. Available: http://cme.mitre.org/index.html

[17] *Malware Attribute Enumeration and Characterization (MAEC)*. Accessed: Dec. 10, 2019. [Online]. Available: http://maecproject.github.io
[18] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, vol. 1. Reading, MA, USA: Addison-Wesley, 1999.
[19] *Virus Total*. Accessed: Oct. 11, 2019. [Online]. Available: https://www.virustotal.com/
[20] N. Lord. (2018). *What is File Fingerprinting?* [Online]. Available: https://digitalguardian.com/blog/what-file-fingerprinting
[21] E. Filiol, *Computer Viruses: From Theory to Applications*. Paris, France: Springer, 2006.
[22] K. Dunham, "A fuzzy future in malware research," *ISSA J.*, vol. 11, no. 8, pp. 17–18, 2013.
[23] V. Roussev, "Data fingerprinting with similarity digests," in *Proc. IFIP Int. Conf. Digit. Forensics*. Berlin, Germany: Springer, 2010, pp. 207–226.
[24] Y. Li, S. C. Sundaramurthy, A. G. Bardas, X. Ou, D. Caragea, X. Hu, and J. Jang, "Experimental study of fuzzy hashing in malware clustering analysis," in *Proc. 8th Workshop Cyber Secur. Experimentation Test (CSET)*, 2015.
[25] R. Rivest, *The MD5 Message-Digest Algorithm*, document RFC 1321, 1992.
[26] M. Colajanni, D. Gozzi, and M. Marchetti, "Collaborative architecture for malware detection and analysis," in *Proc. IFIP Int. Inf. Secur. Conf.* Boston, MA, USA: Springer, 2008, pp. 79–93.
[27] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.
[28] H. S. Nwana, "Software agents: An overview," *Knowl. Eng. Rev.*, vol. 11, no. 3, pp. 205–244, Sep. 1996.
[29] A. Kantamneni, L. E. Brown, G. Parker, and W. W. Weaver, "Survey of multi-agent systems for microgrid control," *Eng. Appl. Artif. Intell.*, vol. 45, pp. 192–203, Oct. 2015.
[30] Z. Guessoum, "A hybrid agent model: A reactive and cognitive behavior," in *Proc. 3rd Int. Symp. Auto. Decentralized Syst. (ISADS)*, Nov. 1997, pp. 25–32.
[31] FIPA. (2002). *FIPA ACL Message Structure Specification, Foundation for Intelligent Physical Agents*. Accessed: Jun. 30, 2004. [Online]. Available: http://www.fipa.org/specs/fipa00061/SC00061G.html
[32] A. H. Bond, "Distributed decision making in organizations," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Dec. 1990, pp. 896–901.
[33] E. Raff and C. Nicholas, "Hash-grams: Faster N-gram features for classification and malware detection," in *Proc. ACM Symp. Document Eng.*, 2018, p. 22.
[34] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Cypider: Building community-based cyber-defense infrastructure for Android malware detection," in *Proc. ACM 32nd Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 348–362.
[35] S. K. Sahay and A. Sharma, "A survey on the detection of windows desktops malware," in *Ambient Communications and Computer Systems*. Singapore: Springer, 2019, pp. 149–159.
[36] M. Belaoued and S. Mazouzi, "A chi-square-based decision for real-time malware detection using pe-file features," *J. Inf. Process. Syst.*, vol. 12, no. 4, pp. 644–660, 2016.
[37] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digit. Invest.*, vol. 24, pp. S48–S59, Mar. 2018.
[38] M. Pietrek, "Peering inside the PE: A tour of the Win32 (R) portable executable file format," *Microsoft Syst. J.-US Ed.*, vol. 9, no. 3, pp. 15–38, 1994.
[39] N. Cressie, "Statistics for spatial data," *Terra Nova*, vol. 4, no. 5, pp. 613–617, 1992.
[40] A. Galton, *Temporal Logics and Their Applications*, vol. 10. London, U.K.: Academic, 1987.
[41] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," *J. Comput. Virol.*, vol. 7, no. 2, pp. 107–119, May 2011.
[42] S. Tisue and U. Wilensky, "Netlogo: A simple environment for modeling complexity," in *Proc. Int. Conf. Complex Syst.*, Boston, MA, USA, vol. 21, 2004, pp. 16–21.
[43] F. Stonedahl and U. Wilensky, "NetLogo virus on a network model," in *Proc. Center Connected Learn. Comput.-Based Modeling*. Evanston, IL, USA: Northwestern Univ., 2008.
[44] I. Sakellariou, P. Kefalas, and I. Stamatopoulou, "Enhancing NetLogo to simulate BDI communicating agents," in *Proc. 5th Hellenic Conf. AI, Artif. Intell., Theories, Models Appl.* Syros, Greece: Springer, Oct. 2008, p. 263.

MOHAMED BELAOUED received the master's and Ph.D. degrees in computer science from the University of Skikda, in 2011 and 2016, respectively. He is currently an Associate Professor with the University of Constantine 1, Algeria, and also a Researcher with LIRE Laboratory, Software Technologies and Information Systems Department, University of Constantine 2. His research interests include malware analysis and detection, intrusion detection, network security, and the Internet of Things (IoT) security.



ABDELOUAHID DERHAB received the Engineering, master's, and Ph.D. degrees in computer science from the University of Science and Technology Houari Boumediene, Algiers, in 2001, 2003, and 2007, respectively. He was a Full-Time Researcher with the CERIST Research Center, Algeria, from 2002 to 2012. He is currently an Associate Professor with the Center of Excellence in Information Assurance, King Saud University, Riyadh, Saudi Arabia. His research interests include network security, intrusion detection systems, malware analysis, mobile security, and mobile networks.

SMAINE MAZOUZI received the M.S. and Ph.D. degrees in computer science from the University of Constantine, in 1996 and 2008, respectively. He is currently an Associate Professor with Université 20 Août 1955-Skikda. His fields of interest are pattern recognitions, machine vision, and computer security. His current research concerns using distributed and complex systems modeled as multiagent systems in image understanding and intrusion detection.

FARRUKH ASLAM KHAN (Senior Member, IEEE) received the M.S. degree in computer system engineering from the GIK Institute of Engineering Sciences and Technology, Pakistan, and the Ph.D. degree in computer engineering from Jeju National University, South Korea, in 2003 and 2007, respectively. He also received professional training from the Massachusetts Institute of Technology, New York University, IBM, and other professional institutions. He is currently a Professor with the Center of Excellence in Information Assurance, King Saud University, Riyadh, Saudi Arabia. He is also a Founding Director of the Wireless Net-working and Security Research Group, National University of Computer and Emerging Sciences, Islamabad, Pakistan. He has over 80 publications in refereed international journals and at conferences. He has co-organized several international conferences and workshops. He has successfully supervised four Ph.D. students and 16 M.S. thesis students. Several M.S. and Ph.D. students are currently working under his supervision. His research interests include cybersecurity, body sensor networks and e-health, bio-inspired and evolutionary computation, and the Internet of Things. He is on the panel of reviewers of over 30 reputed international journals and numerous international conferences. He serves as an Associate Editor for prestigious international journals, including the IEEE Access, *PLOS One*, *Neurocomputing* (Elsevier), *Ad Hoc and Sensor Wireless Networks*, *KSII Transactions on Internet and Information Systems*, *Human-Centric Computing and Information Sciences* (Springer), and *Complex and Intelligent Systems* (Springer).