OMMA: open architecture for Operator-guided Monitoring of Multi-step Attacks

Julio Navarro, Véronique Legrand, Aline Deruyver, Pierre Parrend

Navarro *et al. EURASIP Journal on Information Security* (2018) 2018:6
https://doi.org/10.1186/s13635-018-0075-x

EURASIP Journal on Information Security

RESEARCH Open Access

OMMA: open architecture for Operator-guided Monitoring of Multi-step Attacks

Julio Navarro[1,4*†], Véronique Legrand[2,3†], Aline Deruyver[1,4] and Pierre Parrend[1,4,5†]

Abstract

Current attacks are complex and stealthy. The recent WannaCry malware campaign demonstrates that this is true not only for targeted operations, but also for massive attacks. Complex attacks can only be described as a set of individual actions composing a global strategy. Most of the time, different devices are involved in the same attack scenario. Information about the events recorded in these devices can be collected in the shape of logs in a central system, where an automatic search of threat traces can be implemented. Much has been written about automatic event correlation to detect multi-step attacks but the proposed methods are rarely brought together in the same platform. In this paper, we propose OMMA (Operator-guided Monitoring of Multi-step Attacks), an open and collaborative engineering system which offers a platform to integrate the methods developed by the multi-step attack detection research community. Inspired by a HuMa access (Navarro et al., HuMa: A multi-layer framework for threat analysis in a heterogeneous log environment, 2017) and Knowledge and Information Logs-based System (Legrand et al., Vers une architecture «big-data» bio-inspirée pour la détection d'anomalie des SIEM, 2014) systems, OMMA incorporates real-time feedback from human experts, so the integrated methods can improve their performance through a learning process. This feedback loop is used by Morwilog, an Ant Colony Optimization-based analysis engine that we show as one of the first methods to be integrated in OMMA.

Keywords: Advanced persistent threats, Event correlation, Intrusion detection systems, Multi-stage attacks, Network security

1 Background

Recent cyberattacks are highly targeted and more sophisticated than ever [1]. The economic gains through extortion or stealth can be very attractive for cybercriminals, so they are willing to expend lots of resources to abuse specific victims. Sophistication has also reached global malware campaigns, as we have seen with WannaCry malware, which has recently infected more than 230,000 computers all over the world in a few days[1].

The term *multi-step* is assigned to these complex attacks because they are composed of different steps, either legal or not, with no evident link between them. Therefore, multi-step attacks cannot be described by less than two events [2]. The set of events involved in a multi-step attack is also called an *attack scenario*. It is important to note that individual events which seem innocuous may be part of an attack scenario.

Attack detection has evolved to face these new threats, but a lot of work remains to be done. The execution of a multi-step attack usually leaves traces in several assets in the network and can only be detected from a global perspective, considering the whole attack strategy. Traces can be collected by a System of Information and Event Management (SIEM), a security system able to deal with data coming from heterogeneous sources. A SIEM incorporates mechanisms for analysis and also automatic detection engines [3], generally based on handcrafted signatures coding the behavior of an attacker during a multi-step attack. The detection of threats through the identification of several traces with an attack scenario is

*Correspondence: navarrolara@unistra.fr
†Equal contributors
[1]Laboratoire ICube, Université de Strasbourg, 11, Rue Humann, Strasbourg, France
[4]Unitwin UNESCO Complex System-Digital Campus, Paris, France
Full list of author information is available at the end of the article

called *correlation*. As a result of detection, the SIEM raises an alert, which is sent to the human expert through a computer interface.

Given the complexity of the conception of a SIEM, the global perspective of Engineering Systems brings significant insights. While classical system design focuses on technical details first to elaborate the overall architecture, Engineering Systems are conceived starting with an abstract framework [4]. This is expressed in an architecture meeting the requirements of the system. In a later phase, the defined modules are implemented and the global design can evolve to be adapted to new technical requirements.

Despite the numerous publications proposing automatic multi-step attack detection methods based on event correlation (see Section 3), we have not found in the literature any proposal to combine them in a single system. Moreover, some of them [5, 6] do not even disclose the details about how the detection method works, so the method cannot be reproduced by other researchers. Open research can help the development of more advanced methods able to deal with current multi-step attacks. The challenge is to develop a framework where these methods can work together and around which the exchange of ideas contributing to the development of multi-step attack detection could take place.

In response to this, we propose the architecture of an engineering system called OMMA, Operator-guided Monitoring of Multi-step Attacks, for integration of multi-step attack detection methods working with heterogeneous sets of events. OMMA proposes a framework for merging different detection techniques in order to improve research collaborations and profit from past work. OMMA is one of the architecture proposals developed in the context of the HuMa project. It is directly inspired by HuMa reference architecture [7] and Knowledge and Information Logs-based System (KILS) system [3]. As them, OMMA incorporates real-time feedback from the human expert, which can be used by the integrated methods to improve their performance through a learning process. We consider that the last verdict about what is a threat for the system must be given by a human analyst, whose creative thinking and knowledge about the network allow him to determine the consequences of events marked as malicious by the system.

However, OMMA differs from HuMa and KILS in some aspects:

- Unlike HuMa [7], which is oriented towards threat analysis and forensic investigation, OMMA is focused on real-time detection, taking the approach of classic signature-based SIEM but giving elements for the integration of machine learning algorithms.
- OMMA differs from KILS [3] in that it offers an open architecture for the integration of different detection methods. KILS presents a general cognitive model but not the details about the architecture or included analysis methods.

The main contribution of OMMA is that it offers to the research community an open platform where no matter w hich multi-step attack detection algorithm based on event correlation could be integrated. The name OMMA comes from ancient greek ὄμμα, which means "the eye of heaven" [8]. Apart from the architecture of OMMA, in this paper, we present the example of integration of Morwilog [9] in OMMA, an Ant Colony Optimization (ACO)-based method which was presented in the 2016 IEEE Symposium Series on Computational Intelligence (SSCI).

The rest of the paper is organized as follows. We first present the main challenges on multi-step attack detection in Section 2. In Section 3, we review related work, and we continue in Section 4 with a description about the methodology used in our research. Then, we present definitions of core concepts of the proposed framework in Section 5. OMMA architecture is presented in Section 6. We then explain the Morwilog algorithm in Section 7 and provide experimental results in Section 8. We finish the paper with the discussion in Section 9 and state the conclusions and future work in Section 10.

2 Challenges on event correlation for multi-step attack detection

In this section, we present some challenges of event correlation in the context of multi-step attack detection. We start explaining what is a log in the context of network security. After that, we present a summary of the challenges posed by multi-step attacks, from which we can infer that event correlation is a pertinent approach to detect this type of attacks. We end the section with a brief analysis about classical correlation and its challenges, which constitutes the base of the development of OMMA.

2.1 Log entries

An *event* is "an identifiable action that happens on a device and is recorded in a log entry," according to the Standard on Logging and Monitoring published by the European Commission in 2010 [10]. *Log entries* are also simply referred to as *logs* and are the real manifestation of events in the network. A *log* is generally expressed in plain text, using a language previously defined and dependent on the type of device that registers it [11]. Logs provide a valuable source of information for knowing what is happening in a network. From the point of view of security, the utility of logs resides in the fact that they can be produced by an ample variety of elements connected to the

network, so that information from different origins can be merged to discover threats that would be impossible to discover with information from a single source. It is necessary to translate the logs from different types of source to a common format if we want to use them as a source for event correlation. This process is known as *normalization*.

A log contains both *static*, *dynamic*, and *semantic* information [12]. The static information is composed of the separators and words that depend on the type of log. The dynamic information codes the particularities of each event, usually following certain rules about format. In an already normalized log, the static information is the name of each field and the dynamic one is their value. Finally, semantic information refers to the meaning of the log, to why that log was generated.

2.2 Challenges posed by multi-step attacks

Multi-step attacks are popularly known as *Advanced Persistent Threats (APT)* [13, 14]. We prefer the first name because it is more descriptive about the nature of these attacks and because the multi-step technique, originally linked to targeted attacks, has been already used in global malware campaigns, as WannaCry infection demonstrates. "Multi-step" is no longer an exclusive characteristic of targeted attacks. Anyways, we consider that the term APT has become distorted after its adoption in the industry, as it is frequently used as a marketing technique[2].

The multi-step nature of these attacks brings with it a number of challenges for detection:

- Multi-step attacks cannot be described by less than two events [2].
- An isolated event can be part of an attack scenario even if it seems innocuous.
- Events that belong to the same attack scenario have as a common point that they are the representation of actions leading to a single objective, but this is not necessarily explicit in the events.
- Detection methods count only with a limited amount of information, such as the one contained in events or network packets, to infer attack scenarios.

In event correlation [15], ensembles of *primitive* events are combined to form *composite* events [16]. These latter are a conceptual construct, as the events are observed indirectly only, and they allow inferring the strategy of the attacker. As multi-step attacks cannot be fully detected from the observation of individual events, the need of composite event derivation becomes apparent. Moreover, events involved in a multi-step attack can happen in several assets of different types in the network. The only mechanism present in current networks to get information from all the affected sources is log collection through a SIEM. A SIEM-like architecture can offer a framework where the logs representing events can be collected and normalized to a common format. Then, an algorithm for detection can be applied on the collected data. Event correlation is therefore a convenient approach for multi-step attack detection.

2.3 SIEM and classical correlation

Security systems in charge of log collection and correlation [17] are known as SIEM (System of Information and Event Management). They aim to detect new incidents but also to unify and to reinterpret the alerts generated by other security devices.

Classical SIEMs generally apply *rule-based reasoning* [18], based on identifying the match between static rules and events in the set. Rules are written by an expert according to the nature of composite events corresponding to an attack and, in some cases, to the characteristics of the network where the device is placed. Rules can always be expressed as branches in a tree, with the events as nodes in the shape of a regular expression. Many commercial SIEMs still base their detection mechanisms on rules furnished by the vendor [19] in combination with specific rules crafted by the user.

There exist many ways for evaluating the rules, the simplest one being the linear check of each rule against incoming events. In classical correlation, if the system detects an event matching the beginning of a rule, a detection thread is opened to wait for other events matching the rest of the conditions in that rule. This is done concurrently for all the rules whose beginning is matched by the incoming event. The system triggers an alert if all the components of a rule are matched. For instance, one of the rules of the system can be "generate an alert after 3 failed login attempts against service X." In this case, if a log from service X representing a failed login attempt is detected, the security system opens a new thread which looks for two more failed attempts coming from the same user. If these events occur, the alert is fired. If nothing is found after a certain period of time, the detection thread is closed in order to free the memory of the system. Chains of events with some events in common can be represented in the shape of a tree as shown in Fig. 1. The rule we propose as example could be represented by the sequence {A, B, C}. Each of the components in the sequence would be identified with a failed attempt.

This classical method is used by many correlation systems, such as OSSIM[3] (Open Source SIEM), whose open code has widely disseminated the idea of security event correlation. Inside OSSIM, an event traverses several steps in the form of a log [20], such as assignation of risk score to the target asset or comparison with

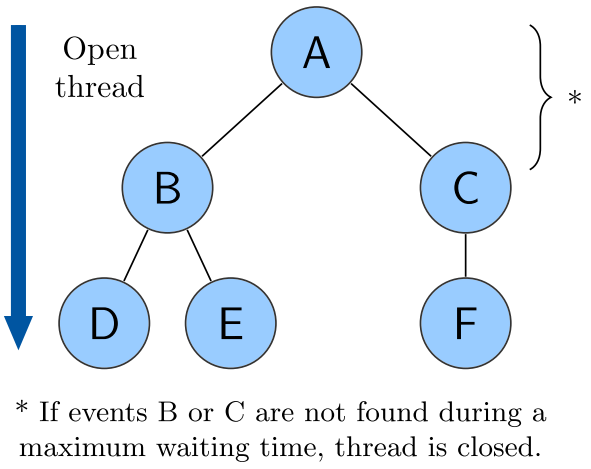

Fig. 1 Example of an event tree for classical correlation with three multi-step attacks

reputation data, among others, before arriving at the correlation engine. The rules defining the sequences of events representing a threat are previously coded by a security expert. Alarms are triggered according to the rule-based reasoning approach, following the process we have just described.

The most positive aspect of rule-based reasoning is that it uses a language easily understood by humans, as rules are expressed in the form of statements "if ... then." Anyways, event correlation needs to be reinvented and improved to be adapted to current scenarios. We have identified several challenges in classical correlation methods:

- Rule definition is manual and requires a lot of efforts by security analysts, not only for their creation but also for updating them, as rules are static.
- The quality of detection depends on the expert's ability.
- Security experts spend too much time looking for incidents in raw records. A greater autonomy of detection systems would reduce this waste of time.
- Rules are just suited to known threats.
- The heterogeneity of events in a network and their great increase in terms of volume and variety in the last few years hinder detection. Sets of rules and parsers become difficult to maintain and control.

OMMA offers an architecture for the integration of methods addressing these challenges. Some issues for solving these challenges are:

- The combination of machine learning algorithms, trained with reference data. We can obtain a better performance in the classification of events as malicious or not if we add up the results from different methods.
- The mix of results obtained in a supervised way with some others coming from unsupervised methods and statistics. Unsupervised learning does not assure the causality implications of links between the events but allows the exploration of unknown sequences and a predefined model does not need to be built [21].
- The evaluation from a human expert, incorporated through a feedback system, as it is proposed by HuMa [7] and KILS [3]. Doing so, causality could be added to the models found in an unsupervised way and the system can learn from experience.

3 Related work

We start this section presenting some security engineering systems working with logs that have inspired us in the development of OMMA. We then review what has been done in the domain of multi-step attack detection through event correlation. Finally, we provide context to Morwilog [9], the ACO-based detection system whose integration in OMMA is proposed.

3.1 Engineering Systems for cybersecurity

Attack detection generally requires the combination of different modules for easing the design and development of the system. Starting from the definition of the framework for later focusing on the technical details allows not to lose sight of the purpose of the system, something really important if it is intended to work within a real network. This is why much work in the field has been done following the approach of Engineering Systems, which considers the definition of the whole system in an abstract way for later proceeding to the implementation.

The main reference in the conception of OMMA is the HuMa reference architecture [7]. It has been proposed in the context of the HuMa project, the same one financing the development of OMMA. Unlike this latter, HuMa is a framework for log analysis, not for attack detection. Its main objective is to deal with massive and heterogeneous sets of logs. It considers the human expert as the main actor in the analysis process. This idea is the one we have adapted to the context of multi-step detection. HuMa organizes the analysis around three layers: the event layer, where individual traces are represented; the context and attack pattern layer, which gathers information about technical requirements of the attacks; and the assessment layer, where information from complex attacks is extracted. HuMa contains a set of analysis methods for the assessment layer. Morwilog has been adapted to be included in both frameworks, HuMa and OMMA.

KILS (Knowledge and Information Logs-based System) is a model also working with logs and it is in the origin of HuMa. It is proposed by Legrand et al. [3] and it is built under the perspective of Big Data, as the amount of logs daily generated in an organization is exceeding the capabilities of current solutions. The authors hold that knowledge from a human expert is fundamental for identifying complex and targeted attacks. Thanks to a feedback loop, this knowledge is introduced into the system, so the attack detection algorithms contained on it can learn and improve their results. The interface with the human is eased by the decomposition of events into abstract concepts. Inspiration is taken from Action Theory, criminal investigations, and bio-inspired methods, but not much detail is given about the implementation of the system. OMMA follows a similar approach in its definition of domains than the levels defined for KILS, called *real world*, *storage*, *inference*, and *expert knowledge*.

Finally, Abreu et al. [6] propose a framework for detecting APTs from log information. Activities are first classified, then ranked according to the priority of generated alerts. The link between activities is found from the set of vulnerabilities an attacker should exploit for accessing the victim. This framework directly comes from industrial research, as all their authors work in security companies. Even if they do not describe the implementation in detail, the paper offers a complete idea about the design of commercial security devices.

3.2 Finding multi-step attacks in datasets of events

The basis of OMMA is the consideration of attacks as an ensemble of events, a strategy composed of different steps. A system would not be able to deduce the objective and nature of the attack considering only one of these individual events. These attacks are denoted as *multi-stage* [22], *multistage* [23], or *multi-step* [2] attacks. Some papers about the topic also refer to them as *attack strategies* [24], *attack plans*, [25] or *attack scenarios* [26]. Lastly, many authors use the name of *Advanced Persistent Threats* (*APT*) [13], a vague denomination which is much used in the industry.

The first public mention of attack strategy as an important point for attack detection is probably in the work by Huang et al. [24]. Taking inspiration from battleground management, they propose a communication protocol between IDS (Intrusion Detection System) agents distributed through the network, with a master controlling their behavior. Agents are responsible for local data analysis and classical intrusion detection. The master analyzes possible strategies, represented in the form of goal-trees, for determining in which kind of event each agent should be focused.

Most of the work about multi-step attack detection focuses on alerts generated by an IDS and considers the construction of attack scenarios through the linking of these alerts. This is the case of the work done by one of the pioneers in the field of multi-step attack detection, Peng Ning [27], one of the prime integrator of prerequisites and consequences in IDS alerts. We consider significant that most of the work in the field of multi-step attack detection do not directly analyze events but alerts generated by a signature-based Intrusion Detection System, looking for scenarios whose bricks are in turn composed of detected attacks. DARPA 2000 Intrusion Detection Scenario-Specific datasets are broadly used as the main data source for evaluating the performance of these systems, sometimes combined with tests over private network data.

OMMA has been conceived to work with heterogeneous events, not only with IDS alerts, which are a type of security event. That is why we focus only in the literature about multi-step attack detection from general events. For this purpose, some specific languages have been developed to allow a security expert to code the behavior of multi-step attacks as a set of events. Two examples are STATL (State Transition Analysis Technique Language), developed by Eckmann et al. [28], and EDL (Event Description Language). EDL was first proposed by Meier [29] and later improved by Jaeger et al. [2]. Basic EDL is built on the intuitive idea of constructing a sequence of nodes for representing the concatenation of different events using a colored Petri net. The idea seems simple but its formal specification includes very innovative mechanisms for defining the rules, such as the use of tokens going through the network of nodes as search agents for indicating which events the system should look for. The automatic derivation of EDL rules has been recently explored by the same laboratory [30]. They automatically extract signatures from log events, arranged in a taint graph. They have evaluated the approach with the simulation of a multi-step attack (see Fig. 2).

The distributed system proposed by Vogel et al. [5, 31] is also based on signatures written in EDL, which are divided in minimal parts and sent to the local agents. Another example of a distributed system applying multi-step attack signatures, expressed in a language different from EDL, is Quicksand [32]. Signatures in Quicksand are expressed as graphs, whose nodes are used by each of the agents for local detection.

Hidden Markov Models (HMMs) have also been used for multi-step attack detection. The HMMs represent sequences of normal events and are derived from a clean dataset of events. If a sequence does not correspond to any of the models, it is considered as anomalous and identified as a potential multi-step attack. This approach is used by Anming et al. [33], who propose the use of the segmental K-means algorithm to create the HMMs. They just work with OS audit data, thus coming from only one source,

```
EVENT UserLogin
{
 PLACES
  Init {
   TYPE INITIAL
  }

  LoggedIn {
   TYPE EXIT
   FEATURES
    STRING username
  }

 TRANSITIONS
  Init(+) LoggedIn {
   TYPE NT_AUDIT_EVENT_4624
   CONDITIONS
    (True)
   MAPPINGS
    [LoggedIn].username=AccountName
   ACTIONS
    warnln("User "+AccountName+" successfully logged in")
  }
}
```

Fig. 2 Example of an EDL signature called *UserLogin* for a simple login onto a Windows system, by Jaeger et al. [2]

but the method could be applied to a set of heterogeneous events if they are conveniently normalized.

Another system based on anomaly detection is the one developed by Mathew et al. [26]. It uses principal component analysis (PCA). They take attack-free data for building a model using PCA and then they project new data on the created clusters so abnormal behavior is easily identified. They consider a sequence of states for the network, each state composed of information from heterogeneous security data, such as IDS alerts, network sensors, events, or others. They select certain features for defining the states of the network in each moment. This method finds anomaly behavior not necessarily corresponding to an attack, so it is eventually able to find the traces of unknown attacks.

Skopik et al. [13, 34] also focus on the abnormal character of attacks. Their security system detects anomalies after learning from a test set clean of attacks. They have developed a whole mathematical framework for defining hypothesis, rules, and anomalies. Each event class is defined by the combination of a mask $\vec{C_m}$, indicating if a field in the event is relevant or not, and a value $\vec{C_v}$, showing if a field is enforced or prohibited for that class.

Giura and Wang [35, 36] propose a model for multi-step attack detection where the stages of the attack are arranged in a layered pyramid. The goal of the attack is placed at the top of the pyramid and the previous steps are distributed in layers. Each face of the pyramid corresponds to a different domain, such as physical, network, application, and user. They implement the model in a detection framework based on correlation signatures, profiles, and security policies.

Finally, Pei et al. [37] propose what they call "attack story reconstruction." They implement HERCULE, a method inspired by relationships in social networks. A list of possible relationships between events is defined as a first step. Then, the relationships are used to create graphs of events. Each edge in the graphs have an assigned weight value, calculated using a quadratic optimization algorithm. The result of HERCULE is a graph containing all the events related to a multi-step attack.

More information about multi-step attack detection references, together with our conclusions about the state of the field, can be found in our systematic survey [38].

3.3 Ant Colony Optimization

Morwilog, the first integrated module in OMMA, is based on Ant Colony Optimization (ACO) [39], a metaheuristic oriented to solve discrete optimization problems through indirect cooperation within a colony of artificial ants. When real ants search for food, they depose pheromones and create trails other ants can follow to the food source. This process of indirect communication through the modification of the environment is called stigmergy [40]. The higher the concentration of pheromones in a path, the higher the probability for an ant to follow it. This leads to the convergence to the shortest path to food. Pheromone evaporation avoids trail stagnation to a sub-optimal path. A complex behavior arises from the individual actions of the ants, which are not able to see the plan of the whole group. We call it a process of emergence. The results obtained by the colony exceed so much the capabilities of an individual ant that it is difficult not to think that there is an invisible score [41].

The first algorithm applying ACO, Ant System, was published in 1991 [42], with the traveling salesman problem (TSP) as an example application, in the context of Dorigo's Ph.D. thesis [43]. Application of ACO to TSP is fairly straight, as routes can be directly translated to the artificial trails of ants. Since then, it has been successfully applied to many NP-hard optimization problems such as optimizing traffic control signals [44] or scheduling a galvanizing line [45].

An interesting version of ACO is the *Hommilière* (Manhill) system developed by Valigiani [46] in his Ph.D. thesis. It was created to be applied in an e-learning platform for recommending the best learning path for each user, according to the results obtained by other students and the previous students' results. As the number of users is big enough (more than 150,000 in the tested environment), each of them can be associated to an ant. The ant goes through the different lessons, arranged as a graph, depositing pheromones according to the student's success at each step [47].

Independent of its results or its real applicability to e-learning, the most relevant contribution of this work to ACO metaheuristic is the idea of using an element of the real world associated to each ant, instead of generating a base population of artificial ants. This leads to a different point of view in ACO and brings the possibility of incorporating the complexity of natural processes to the generation of ants. Other works have proposed that each ant simulates a real entity, as Mahanti et al. [48] do for simulating attackers in a honeypot environment, but the Manhill algorithm is the first one, as far as we know, directly associating the ants to real-world entities.

The bibliography concerning ant algorithms includes plenty of papers trying to give answers to the problem of network attack detection. However, the only ant-based work we know addressing the problem of multi-step attack is Janus, a model developed by Zhang et al. [49]. It is based on Partially Observable Markov Decision Process and it is two-sided, as it has a double point of view, that of the attacker and that of the defender. Based on this representation, the objective is to search for attack paths with minimum action cost in the case of the attackers and to highlight the most critical points for a successful attack in the case of the defenders.

Several ACO-based papers related to security directly translate the ant metaphor to software agents moving through the network [50, 51]. Nonetheless, if we consider the analysis of events in a central location, as we do in this paper, the most widespread technique is anomaly detection through the clustering of network data [52–54], in some cases combined with supervised methods such as SVM [55]. There are also some approaches introducing fuzzy systems [56] or relying on the importance of distributed clustering [57]. Other examples can be found in [58]. The limitations of all these centralized approaches are that they consider an attack as characterized by only one event and they do not consider the multi-step attack perspective.

4 Methods

The development of OMMA has the objective of providing an operational and open architecture for the detection of multi-step attacks with human assistance. The ideas reflected on its design are based on three axes: our experience in security research, our knowledge about attack detection methods used in the industry, and the study of bibliography about multi-step attack detection. The two main goals of OMMA are the following ones:

- One of our goals is to make OMMA an *open system*, in the sense that its components should be modular and other researchers could develop new detection methods or adapt existing ones for including them into the system. Nowadays, event correlation systems are closed and not very customizable. Commercial ones are black boxes which only allow the inclusion of custom static rules, adapted to customer environment. Users cannot include any new machine learning detection method or any custom heuristic mechanism. In a competition to keep their know-how safe, security companies do not even disclose the name of the kind of methods they use, let alone expose the details of their operation. On the other end of the spectrum, most of the systems proposed by public research are oriented towards a specific method of detection and they do not provide such a modularity.
- Another prerequisite is to keep the *human in a central role* during the detection process [3, 7]. In our experience, determining the malicious nature of a set of events is in most of the cases only within reach of a security expert's creativity. Full automation in attack detection is far from being reached. As the expert is outside the network dimension, he or she can check the consequences of the supposed attack using external mechanisms such as testing the availability of services or interviewing the personnel about actions outside the scope of logging facilities. The possible forms of network threats are so ample that security experts usually have to deal with problems in real time. The idea is to provide the specialist with tools for easing the detection with the minimum waste of time.

With these objectives in mind, we have started to conceive OMMA as a full architecture covering the entire chain for detecting and mitigating multi-step attacks. We have begun with the development of Morwilog [9], the first module to be integrated in OMMA using the

feedback loop. Morwilog uses an ACO paradigm mainly inspired by the Manhill algorithm [47]. We wanted to apply the algorithm developed by Valigiani to attack detection, something that has never been done. As we said in Section 3, ACO has been extensively applied to attack detection, but we do not know any work where there was a relationship between real entities and the creation of new ants.

Once Morwilog is built, we need a dataset accomplishing a certain number of characteristics for testing the system:

- Firstly, the feedback provided by the operator has to be simulated during the experiments. We assume the human expert is infallible so no mistakes are introduced in the system. For simulating this, we need the data to be labelled[4], so the evaluating system can know if an event is part of a multi-step attack or not.
- Another requisite of the dataset is that events should be sent by a broad range of devices. This is the case in real networks, where SIEM devices are able to deal with any kind of events at the same time. Moreover, most of the attacks we want to detect have to be detected through their traces in different locations of the network.
- The final requisite is that attacks injected in the dataset have to represent a multi-step strategy. Single-step attacks can be detected by specific security systems without the need for processing events from different sources at the same time.

We then have to find a labelled dataset of heterogeneous logs where attacks are represented as being composed of several events. During our research, we have not been able to find an up-to-date dataset meeting all these requirements. Because of that, we use an artificial dataset for this initial implementation of the system. The details of this artificial dataset are explained in Section 8.

5 Preliminary definitions

The proposed engineering system is based on a set of concepts which are worth being rigorously defined to ease the explanation. Defining a formal framework for working with logs does not only allow us to use general mathematical concepts, but also to establish a common notation that could be used by the rest of the research community.

5.1 Event operations

OMMA uses the events generated by the devices in the network as data source. An *event* $\boldsymbol{e}$ is the record of an action happening in a given device. It can be considered as an entity in the *set of all possible events* $\mathbb{E}$. We can view the event as a finite set of identifier/value pairs containing information about certain aspects of the event. The event is then expressed as a tuple of components id_n/v_n, so $e = \{id_b/v_b, id_c/v_c, \ldots, id_a/v_a, \ldots\}$. This abstract construction is represented as a log in the context of a network. The *identifier* $\boldsymbol{id_n}$ defines a unique meaning for each value $e\,(id_n) = v_n$ in the event e. Thanks to the identifiers, it is easy to find a correspondence with other events. They determine the meaning of each of the fields we can find in a log. They should be universal and easy to understand, so any kind of log can be merged with others generated by different devices after proper mapping. An identifier unambiguously defines the type of *value* $\boldsymbol{v_n}$, which could be a real number, a string element from a finite set, or an IP address, among others. We can find identifiers as "source," "timestamp," or "action," for instance. Basically, the identifier is the static information of the log and the value is the dynamic one, as it was defined in Section 2. The semantic information can be coded as additional identifier/value pairs.

When working in a network, we can assume we have a *set of events ordered in time*. This can be expressed as $\boldsymbol{E = (e_1, e_2, \ldots, e_{N_E})}$, with $\boldsymbol{N_E}$ the *number of events*. From all the collected events, only a subset is used for attack detection, as duplicated events are grouped or discarded and basic filtering is made in the Collector, as described in Section 6. We call this subset $\boldsymbol{E_{in}}$.

The sequence $s = (s_0, s_1, \ldots, s_{L_s}) = (e_l, e_n, \ldots, e_q, \ldots)$, with $e_l, e_n, e_q \in E$ and $q > n > l$, defines a relationship of correspondence between $L_s + 1$ events from a set E. The *set of all possible sequences* in the set of events E is denoted by $\boldsymbol{S_E}$.

These sequences can potentially represent a multi-step threat. The set of all *sequences representing a threat* is denoted by $\boldsymbol{A_E}$, with $A_E \subseteq S_E$. The task of OMMA, through the algorithms integrated on it, is to find a sequence $s \in A_{E_{in}}$. We define $\hat{A}_{E_{in}}$ as the set of alerts in E_{in}. That means $\hat{A}_{E_{in}}$ is the set of sequences marked by OMMA as threats but where we can find a proportion of false positives. In an infallible system, $\hat{A}_{E_{in}} = A_{E_{in}}$.

We need a way to define a subset of events through an ensemble of constraints, to be able to work with several events without the need for listing them all. For this purpose, we use the concept of *abstract representation* $\boldsymbol{e^*}$. We can see e^* as being composed of rules $e^*\,(id_n)$ to apply to the different components of the event for a finite set of identifiers id_n. The most simple rule is the equality between elements. For example, we can define a rule to refer with an abstract representation to all the events with the field "protocol" equal to "HTTP." However, other relationships could be defined, such as numerical intervals (e.g., "connection number" higher than 650), discrete ranges (e.g., "destination" in the IP address range 175.68.22.0/24), sets of values (e.g., "port" is 80 or 443), or complementary definitions (e.g., "user" is *not* "admin").

We call $\mathbb{E}^*$ the *set of all the possible* $\boldsymbol{e^*}$. The set of all the possible abstract representations of event e_i is denoted as $\boldsymbol{E_i^*}$. E_i^* can be finite or infinite depending on the attributes of the fields represented in e_i. We also define the *operator* $\odot$, which operates between an event e_i and an abstract representation e^* and returns the information about the matching between the two. More precisely, $e_i \odot e^* = 1$ if $e^* \in E_i^*$, while $e_i \odot e^* = 0$ if $e^* \notin E_i^*$. Considering the vision of e^* as a set of rules as we just did after the definition, $e_i \odot e^* = 1$ only if $e_i\,(id_n)$ agrees with the rule defined in $e^*\,(id_n)\ \forall\ id_n$ represented in e^*.

5.2 The event tree

An important concept for OMMA, used by detection methods such as Morwilog, is the *event tree*. Each of its nodes contains an abstract representation which could match with a set of events, as it is shown in Fig. 3. Each tree is unambiguously defined by its root node, which represents the first event of the set of suspicious sequences contained in the tree. This concept is directly extracted from the mechanisms followed in classical correlation (see Section 2 and Fig. 1). We can always represent a set of multi-step rules in a tree, with each node as the abstract representations of a set of events. Once a suspicious event is identified, the tree whose root node matches this event is identified and the system waits for one or several events matching the root node's children. For not wasting the resources of the system, a maximum waiting time is defined, after which the analysis thread is closed even if there are no results. The process of matching and waiting continues from the top to the bottom of the tree until the end is reached and a suspicious sequence is returned, generating an alert. The nodes are then distributed in different levels, each of them representing events later in time as we go down through the branches of the tree.

We should not confound this with the attack trees, extensively used in the literature [59] but where the root represents the final objective of the attacker and paths are sequences of exploitable vulnerabilities.

More rigorously, each tree δ_j is represented by an ensemble of nodes κ_n, so $\delta_j = \{\kappa_0, \kappa_1, \ldots, \kappa_{N_\kappa^j}\}$, with $\boldsymbol{N_\kappa^j}$ the *total number of nodes in tree* $\boldsymbol{\delta_j}$ in a given moment and $\boldsymbol{\kappa_0}$ the *root node*. The *set of all possible nodes* is denoted by $\mathbb{K}$.

Nodes $\boldsymbol{\kappa_n = (e_n^*, F_n, \kappa_n^{(\alpha)})}$ contained in a tree are formally composed of three elements:

- An abstract representation of an event $e_n^* \in \mathbb{E}^*$.
- A set F_n of N_f children $\kappa_m \in \delta_j$.
- A pointer to its ancestor $\kappa_n^{(\alpha)} \in \delta_j$. This is not strictly necessary for the definition of the tree, but it is very useful if we work on the tree with a system that needs to propagate information from the deepest node of the tree to κ_0, as with ant pheromones in Morwilog.

On the element e_n^*, we can use the operator $\odot$ previously defined. With this operator, a system could look for events matching e_n^* in a dataset. For easing the representation, we extend the operator $\odot$ for working between nodes and events, so $\boldsymbol{e \odot \kappa_n = e \odot e_n^*\ \forall\ \kappa_n \in \mathbb{K},\ e \in \mathbb{E}}$.

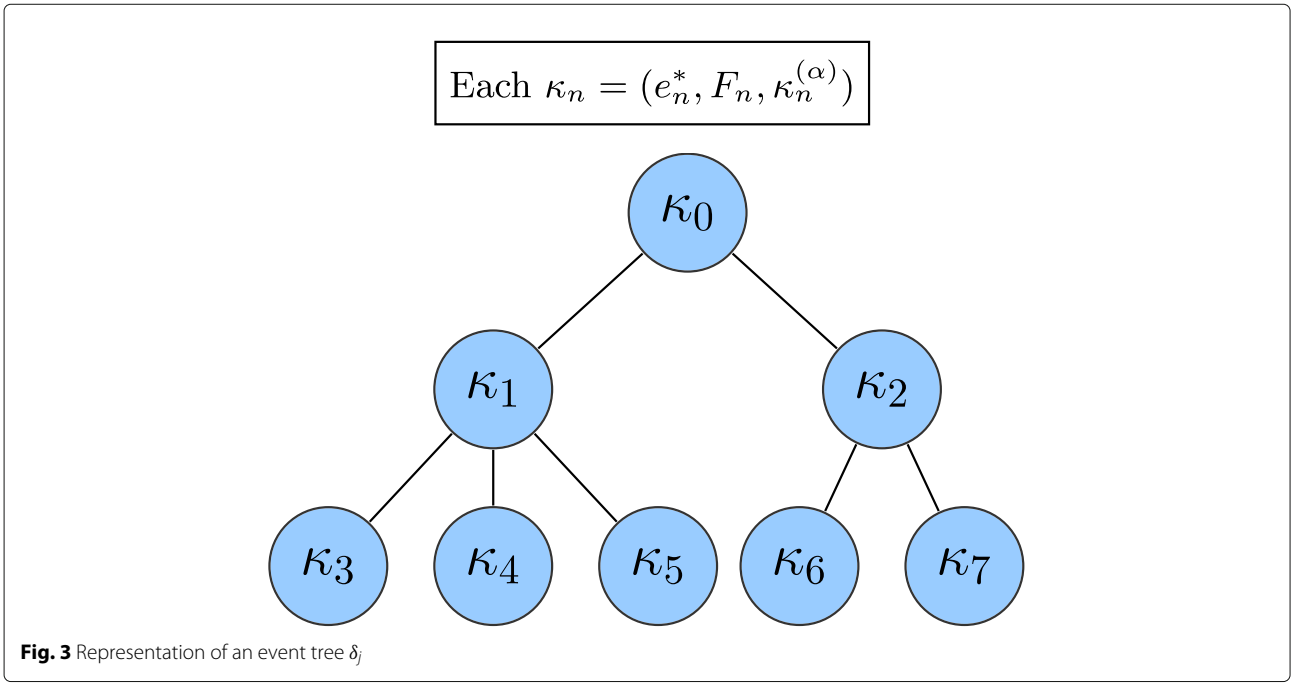

Fig. 3 Representation of an event tree δ_j

If we want to work with heterogeneous events, they should be normalized to a common format before applying the operator, so the components represented in e_n^* mean the same in terms of security independently of the event's origin.

6 System architecture

OMMA engineering system is oriented towards multi-step threat detection through event analysis. This orientation has been chosen as events can be collected from a broad range of devices, irrespective of whether they generate visible IP traffic or not. We have considered a strong presence of the human analyst, who has to verify the alerts and gives some feedback to the system about what is or is not a real attack, called *Fb* in some of the diagrams. The contribution of a human expert is still needed for detection [3, 7], no matter how effective automatic methods used nowadays can appear.

OMMA is within the framework of Artificial Immune Ecosystems (AIE) developed by ICube Laboratory, in Strasbourg (France). This kind of systems, inspired by immune system mechanisms, propose a continuous cycle of detection, analysis, and reaction against external anomalies. AIE also proposes the definition of a detector's life cycle, ranging from their design to the application, plus the learning phase. Apart from multi-step attack detection, it has been previously applied to anomaly detection in time series [60].

In Fig. 4 we can see the global architecture of the system. Defining an architecture is one of the most important steps in the definition of an Engineering System [4]. OMMA architecture is divided in different modules, considered as black boxes for the rest of the system. We can group them in four domains, according to their purpose and place in the analysis chain: *Connection*, *Analysis*, *History*, and *Knowledge*. These domains are compliant to levels in the KILS model [3].

The modules are briefly introduced here and explained in more detail in the following subsections. Each one has a unique two-letter code for better referring to the modules in a diagram.

- *Connection*: communication with the network
  - *Collector (CT):* collection, normalization, and pre-filtering of events
  - *Actuator (AC):* execution of automatic defensive actions in the network
- *Analysis*: automatic detection and learning
  - *Matcher (MT):* signature-based attack detection
  - *Classifier (CS):* detection of suspicious events using supervised and unsupervised methods
  - *Orchestrator (OR):* coordination of the whole system and application of the feedback loop
- *History*: archive of raw and processed data
  - *Logger (LG):* event and alert storage and forensic search
- *Knowledge*: management and storage of information from human and machine experience
  - *Advisor (AD):* management of expert data
  - *Visualizer (VZ):* visual representation for the operator

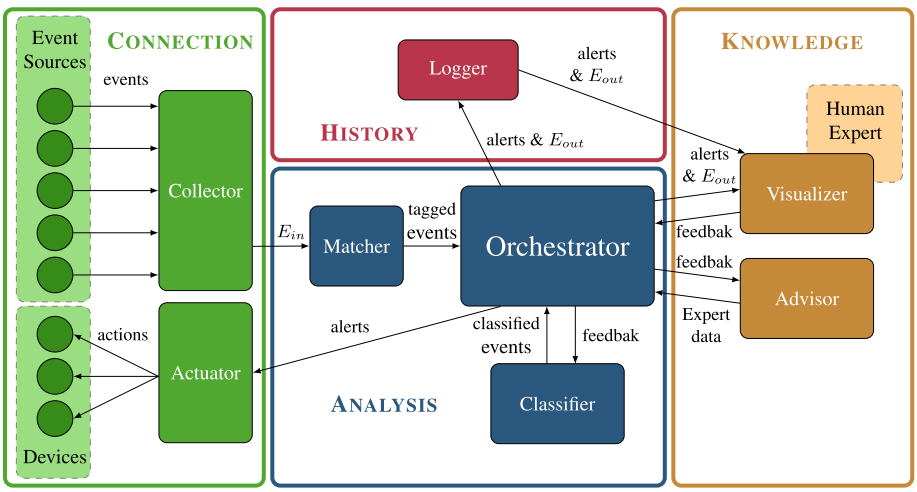

Fig. 4 Global architecture of OMMA (Operator-guided Monitoring of Multi-step Attacks). E_{in} = input events filtered by the Collector. E_{out} = enriched events

6.1 Connection

In the domain called Connection, we find modules acting as an interface between OMMA and the network, the Collector, and the Actuator.

6.1.1 Collector (CT)

Before being able to analyze the events, it is necessary to extract the information contained in the logs. This is the task of the Collector in OMMA. The difficulty resides in being able to process the events coming from different kinds of sources. Each source uses a specific log format defined by the vendor, which forces the Collector to have different parsers adapted to these formats. Parsers map the fields from original logs to the fields used by the system, translating the information into a *normalized* format common to all the sources.

In the industry, parsers are usually manually developed by experts familiar with the log format used by each device. Vendors make these parsers available to their customers in the form of plugins, which can be integrated into the same machines where the correlation system is installed, into a different server, or even into the device generating the logs itself. They also include some tool or language so customers can develop their own custom parsers. The problem of manual creation of parsers is that technologies change and so do log formats, so their operation should be checked after each system update.

Even if there are some interesting alternatives for automatic parsing, such as the ones based on NLP (Natural Language Processing) [61] or the LTE (Log Template Extraction) method [11], they cannot be applied to all kind of logs and need further development. As the aim of our system is detection and not normalization, we consider a regular custom normalization, adapted to the format of log registered by each of the elements in the network. The Collector could even be replaced by a commercial event sensor if it has a well-developed interface to retrieve the normalized logs.

Log collection is made by several daemons running in parallel, called *captors*. There are two general kinds of *captors*, one push-based and another pull-based. The first one is for log sources able to send the logs to the Collector by themselves, using *syslog* [62] or a similar mechanism. On the other side, we need pull-based daemons for devices which need to be periodically requested to send the logs (e.g., Microsoft Windows operating system).

The Collector could also apply basic filtering, discarding events we do not want to include in the analysis and merging duplicate ones. The output of this module is a set of normalized events, which come out from it in real time.

6.1.2 Actuator (AC)

The Actuator is the module responsible for performing prescribed actions in the network so as to stop or prevent the development of an attack. The actions performed can be very diverse, from applying a firewall rule to disconnecting a server from the Internet.

This is the most difficult element to develop in this architecture. Being able to determine an automatic association between a defensive action and the nature of an attack is not an easy task. Even a security expert may not directly have the answer and has to try different methods before arriving at blocking the attack. Anyways, the Actuator could play an important role in the final system, and its implementation could be optional and selective, only taking part for attacks against which automatic defensive mechanism can be applied.

6.2 Analysis

The modules which belong to the Analysis domain perform the automatic detection of multi-step attacks. The static signature-based detection is done in the Matcher, while the Classifier is in charge of applying machine learning or heuristic techniques. The central coordinator of this domain, and of the whole OMMA system, is the Orchestrator.

6.2.1 Matcher (MT)

The Matcher applies classical correlation and attack detection based on signatures, so well-known threats can be rapidly detected. Events composing them are tagged as malicious, so the rest of the modules know about their nature as threats. They are still included in later analysis for checking if they are part of a more complex multi-step attack scenario. The resulting tagged set of events is called E_{tag}.

This module can be implemented by any event-based attack detection system. The only constraint is the input and output format, which has to be adapted to be understood by the rest of the modules. Tags can be simply considered as an additional field in the logs.

6.2.2 Classifier (CS)

This module has the task of classifying the events in terms of security. Sets of events clearly corresponding to good behavior have to be excluded from the analysis. Suspicious ones, with all those related to them, have to be preserved for being further handled by the Orchestrator. Detection of threats can be reduced to a mere classification between good and bad events.

Classification can be made using machine learning in a supervised or unsupervised way. The idea is to combine the power of several algorithms for later obtaining a verdict in the Orchestrator. This bank of methods is adaptable, and any new event-oriented algorithm could be incorporated to the analysis. New algorithms should adapt their input and output formats to the ones defined in the implementation of the system. Not every algorithm

is valid for every type of data, so a pre-selection is made in the Orchestrator.

Once the classification mechanisms are defined, events can be classified as they are collected and grouped into batches to ease the analysis.

6.2.3 Orchestrator (OR)

After the Matcher, events are sent to the Orchestrator. This is the central module of the system, the one coordinating the detection process and making the final decision about which set of events is a threat for the network. It takes the information given by the Classifier and the Advisor about tagged events coming from the Matcher (E_{tag}), evaluates it, and sends the results to the Visualizer, the Actuator, and the Logger. It also gets feedback from the human expert through the Visualizer and sends the results to the detection methods.

The Orchestrator is in charge of evaluating the results returned by the different methods included in the Classifier. For that, it follows a voting process. It knows the nature of each involved algorithm and its degree of reliability, so it focuses in one or another depending on the kind of data to be analyzed. This voting system gets feedback from the false positives and false negatives as they are introduced by the human analyst through the Visualizer.

The Orchestrator also coordinates the delivery of data from the Advisor to the different methods in the Classifier, periodically requesting the information needed by each of them. It is important to note that the Classifier is just in charge of the algorithm execution, but the decision about which ones are applied to which set of events is made by the Orchestrator.

Once a verdict is issued about the nature of the events, alerts $\left(\hat{A}_{E_{out}}\right)$ are sent to the Visualizer, to the Actuator, and to the Logger, which also receives the events, enriched during the process (E_{out}). Alerts generated in this phase are arising from the events which are clearly labelled as malicious.

6.3 History

The domain in charge of storing historical data is called History, and in our design, it includes just one module, the Logger.

6.3.1 Logger (LG)

This module manages the historical data of the system, both logs and alerts. Periodically, a routine is executed for writing to disk logs and alerts on memory. It happens that for some algorithms in the Classifier, events come too fast to be analyzed. The storage in the Logger allows applying these algorithms later, during forensic investigation.

The Logger also includes routines for log and alert searching, so the analyst can do forensic investigations through the Visualizer.

6.4 Knowledge

The Knowledge domain is where the feedback from the human expert is introduced. Expert information can come through two different ways. The Advisor stores expert data from past experiences, while the Visualizer takes input from the human in real time, also providing a console output showing events, alerts, and the state of the system.

6.4.1 Advisor (AD)

The Advisor is the module in charge of managing and storing knowledge on events and attacks, which is either manually introduced by security experts or automatically learned by machine learning algorithms. Such knowledge contains, for instance, the event trees used by Morwilog. The Orchestrator distributes this information to the corresponding methods in the Classifier. Moreover, it receives feedback after positive or negative detection, so expert data is updated and improved.

One of the aims of this module is to unify the data format used by correlation algorithms as much as it is possible, not only inside our security system but also in the communication with external systems. Attack description has to be legible both for humans and machines, as well as easy to exchange between modules. So far, the event trees used by the system are general enough for being generated by any tool, even in real time during the analysis.

6.4.2 Visualizer (VZ)

The Visualizer is the module that represents the information of alerts and logs to the human user. This module controls the presentation of the data generated by the other modules but also transmits the input from the user into the feedback loop, so pheromones in Morwilog can be modified and the algorithms in the Classifier can identify false and true positives and learn and modify the classification methods for improving the detection rate. New alerts are generated and represented in a console so that the human expert can progressively evaluate and mark them as attacks or not. The Visualizer is also provided with tools for doing forensic investigation on events and alerts stored in the Logger.

6.5 Some comments on modularity

The architecture of OMMA is totally modular. Each module has been conceived for being independent from the others, and its functions can be replicated in more than one instance. On the other hand, some modules can be also grouped in the same network element. We follow here the perspective of Artificial Immune Ecosystems (AIE) [60], which conceives a design where centralized elements are combined to distributed ones, according to the needs of the implemented system.

We list below some reasons why modularity is interesting for this system.

- *Different types of event can be processed separately.* We can decide to have certain modules dedicated to specific tasks. We can have, for instance, several Classifiers, each one with a set of algorithms dedicated to a specific type of event. This can also ease the combination of modules developed by different research units.
- A *hierarchical design* is possible, with an instance of one module acting as master and some others as slaves. This is important if we deploy the system in a large network. We can, for example, have an Orchestrator master controlling some other ones acting as agents, in different areas of the network or remote sites.
- We can do *load balancing* between different instances of the same module. Doing so, we can combine small modules for having a processing power equivalent to a bigger unit, therefore saving cost. For example, we can think of some instances of the Collector module working together for processing a high volume of events.
- A module can be doubled for *High Availability*. If one instance fails, the other one can automatically take its place, avoiding system interruptions. The most suitable element for High Availability is the Logger, as it stores sensitive information which is worthy to be replicated, avoiding data loss in case of failure.
- It can lead to *economies of hardware or design*, as several modules can be grouped into the same device, specially the ones belonging to the same domain. For example, we can have an instance of each of the modules belonging to the Analysis domain, the Orchestrator, the Classifier, and the Matcher, in the same box.

In Fig. 5, we present the diagram of a possible implementation of the system in a network, where we can see how the idea of modularity is applied. We have a Collector for each of the subnetworks A and B. Both of them report to the same Matcher, and only one Actuator works for both subnetworks. Subnetwork C counts with its own Actuator and Matcher but also with two active Collectors doing load balancing. The Orchestrator, the Classifier, and the Advisor are all located in the same device. This allows not sending the data they exchange out in the network. Moreover, shared hardware eases the implementation and reduces delays in information transmission. The Visualizer is implemented as a device by its own. Finally, we find two Loggers in High Availability configuration (HA) forming a cluster. All the information is copied into the two devices, so it is well preserved in case one of them would stop working.

7 Morwilog

The most important element in our engineering system is the feedback loop, as it is responsible for the learning process. Thanks to the feedback from the human experts, the system improves its multi-step attack detection rate. Here we present the Morwilog system, which directly uses the feedback data retrieved from the expert [9]. Once suspicious events have been selected, Morwilog tries to find the links between the events belonging to the same multi-step attack using the historical data of event trees. The database of event trees can be built through training data or manually from experience. A mechanism of random generation of trees, which is used in the experiments presented in this paper, is also integrated in Morwilog, so if there is no tree corresponding to an input sequence of logs, a new event tree is built.

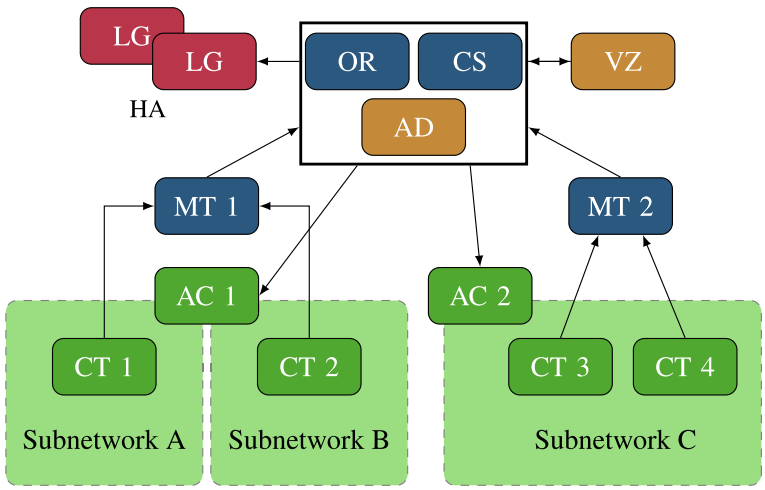

Fig. 5 Example of an implementation of OMMA where we can see its modularity

7.1 From Manhill to Morwilog

Morwilog is based on ACO [39] and, especially, Manhill [46]. The name Morwilog comes from the prefix meaning "ant" in Proto-Indo-European [63] ("morwi-" in latin characters). The Proto-Indo-European is a theoretical reconstruction of the common ancestor of the Indo-European languages [64], the supposed mother of English or Latin, among many others.

The most relevant idea behind Morwilog is that the generation of the artificial ants, called *morwis*, takes places from the detection of real entities, the events. This idea directly comes from the Manhill algorithm, where the real entities are the students connecting to the e-learning platform, each of them represented as an artificial ant. The real entity generating the artificial agent does not have to be necessarily a living one, but it can be anything generated by such an entity and therefore unpredictable, such as the events.

The generation of *morwis* from the events arriving at the system follows the same principle as the generation of analysis thread in a classical correlation system. Each *morwi* follows a path in an event tree, a decision tree with representations of events as nodes (see Section 5). Pheromones are added between each pair of nodes on this path. These pheromones, or counters, are modified by the *morwis* according to their success in detecting valid sequences. The sequences returned by the *morwis* as representing an attack are sent to the Visualizer and they are evaluated by a security expert, which reviews alerts as is performed for instance in a SOC (Security Operations Center).

To implement Morwilog as part of the OMMA architecture, we have first developed an instance of the Collector (see Section 6 and Fig. 4). Logs used are already preprocessed and their fields are normalized [65]. The Collector reads directly from a log file and then iterates through the logs, which are ordered in time. A function called *LogSubsetIterator* is the one in charge of the consecutive arrival of logs to Morwilog.

All the code has been written in C++ following an object-oriented programming paradigm. There are classes corresponding to the event trees (*AttackTree*) which are grouped in objects representing a forest (*AttackForest*). *Morwis* are also objects, which are created with the arrival of each log and destroyed after they finish to go through the tree and the level of pheromones is updated. The class responsible for the execution of the algorithm and the coordination of the *morwis* is called *Loghill*. A simple instance of the Classifier has been also developed. It separates the logs according to the maximum time between logs T_{max}. To summarize, the flow of the implemented program starts with the collection of logs from a file by an instance of the Collector and ends with the execution of Morwilog by the module *Loghill* after a classification of logs in time windows.

7.2 The stigmergic scenario

The *event tree* we have defined in Section 5 is the scenario where the pheromones from the *morwis* are deposited and therefore where the stigmergic process takes place. When an event e_i arrives at the Morwilog system, a *morwi* is generated and it starts its search in a node κ_0 matching e_i (Table 1). The search of the following events in the sequence takes place during a maximum time T_{max}, as the system does not have an unlimited amount of resources.

The arcs connecting two nodes κ_n and κ_m have a particular level of pheromones $\tau_{n,m} \in \mathbb{R}$. The pheromones give information to the *morwis* about which path they should choose with higher probability. If κ_m is a child of κ_n and it has further children, the level of pheromones $\tau_{n,m}$ associated to the path to κ_m is the sum of the elements $\tau_{m,l} \; \forall \; \kappa_l \in F_m$, so pheromones in F_n always depend on values in the deepest nodes.

It is important to note that the way we implement the set of event trees in the final system has to take into account possible limitations, such as system performance or structural simplicity. In the implementation of Morwilog, we have arranged the event trees in a finite set $\Delta = \{\delta_1, \delta_2, \ldots, \delta_{N_\delta}\}$ called *forest*. When a *morwi* is created, one of the N_δ trees present in the system at that moment is selected according to the match between the event generating the process and e_0^* in node κ_0 of that tree.

7.3 Pheromone evolution

As in classic ACO, the level of pheromones evolves according to the results returned by the *morwi* after traversing the tree. The level of pheromones (τ) can be incremented for reinforcing a path leading to a real threat

Table 1 Nodes in the example represented in Fig. 7, containing the propagation steps in WannaCry

Level	Node	Matched event	Origin
0	κ_0	Unsuccessful HTTP request from a host, called A, to long domain name	Proxy
1	κ_1	Successful connection in port TCP 445 from A to any another endpoint (B) in the local network	Internal firewall
1	κ_2	Successful HTTP request to the same domain name as in κ_0	Proxy
2	κ_3	SMBv1 communication between A and other machine using command "transaction2_secondary"	Internal firewall
2	κ_4	Malformed SMB headers for the NEGOTIATE PROTOCOL REQUEST from A to B (CVE-2009-3103)	Endpoint B
2	κ_5	SQL injection alert coming from A	IDS

or decremented for penalizing one resulting in a false alarm. This evolution of the level of pheromones takes place once the sequences have been analyzed and tagged by a security expert through the Visualizer. New attacks are identified from their consequences in the real environment and the network's normal functioning. The *morwi* only generates an alert if the level of pheromones associated to the path is above τ_{min}^{atk}, defined as a parameter of the system.

The amount pheromones incremented is called $\Delta\tau^+$, while the decrement is $\Delta\tau^-$. We have that $\Delta\tau^+ > 0$ and $\Delta\tau^- < 0$. If we consider $\tau[n]$ as the level of pheromones after update n, with $n \in \mathbb{N}$:

$$\tau[n+1] = \tau[n] + \Delta\tau^{+,-} \tag{1}$$

For avoiding stagnation, a mechanism of pheromone evaporation is necessary. This avoids the *morwis* to concentrate themselves around sequences that have become well-known threats. The detection of these sequences can be made using pattern matching techniques, so new threats discovered by the *morwis* should be translated into static rules and sent to the Matcher.

The evaporation consists in the decrease of the previous level of pheromones by an evaporation rate ρ, with $\rho \in \mathbb{R}$ and $0 < \rho < 1$. The equation for calculating the level of pheromones after evaporation is:

$$\tau[n+1] = (1-\rho) \cdot \tau[n] \tag{2}$$

The evaporation mechanism is only applied to nodes in the same branch as the chosen path for not penalizing rare events. If evaporation was applied to every link in the tree, there would be many nodes that cannot be chosen since there is no event matching them. This would not mean that they could have a high possibility of leading to an attack. Evaporation is always applied before incrementing or decrementing the level of pheromones.

More generally, modification of pheromones is applied directly only to nodes with the same ancestor as the last node in the path returned by the *morwi*. Modifications are then propagated up in the tree to the root node κ_0, added up for preserving the definition of pheromones in every F_n. This is the reason why we have defined pointers to the ancestors in the tree nodes.

There is here an important variation with regard to the classic ACO literature, where both increment and decrement are constant and independent of the level of pheromones. Combining this with the evaporation leads to a variation of pheromones whose absolute value decays as the system evolves. However, as we still want a higher decay in pheromone evolution for strongly penalizing badly chosen paths, we have introduced a dependence on $\tau[n]$ in $\Delta\tau^+$ and $\Delta\tau^-$, making $\Delta\tau^+$ with the shape of a Gaussian function and $\Delta\tau^+ = -\Delta\tau^-$.

$$\Delta\tau^+(\tau[n]) = \Delta\tau_0^+ e^{-\frac{(\tau[n]-\tau[0])^2}{2w^2}} \tag{3}$$

The value $\tau[0]$ is the initial level of pheromones, and it is fixed as a parameter. This is the value to which the pheromones of all the node-to-children links are initialized when a new tree is created. The Gaussian function is centered to $\tau[0]$ because we want the higher change right after a new node joins the tree. $\Delta\tau_0^+$ is the increment of pheromones when $\tau[n] = \tau[0]$, and w is a parameter determining how spread the increment function is.

In classical ACO, the difference between consecutive updates of the level of pheromones is smaller in later stages of the execution, as the level of pheromones is further away from the initial value. This ends up in the convergence to an upper limit of pheromones in absolute value, as it has been proven in [39]. Increment in Eq. 3 should also converge to a maximum value. We can imagine an infinite succession of increments in one of the arcs during the execution. After each step, we have a new value of pheromones for the arc:

$$\tau[n+1] = (1-\rho) \cdot \tau[n] + \Delta\tau_0^+ e^{-\frac{(\tau[n]-\tau[0])^2}{2w^2}} \tag{4}$$

Calculating the limit of this recursive function when we have an infinite number of steps n is not as trivial as in classical ACO. We can prove this upper limit exists for certain values of the parameters just running the recursive function during a high number of pheromone updates. This prevents the system to unlimitedly favor an arc.

In the curve "Increment" of the diagram in Fig. 6, we represent the evolution of pheromones in Eq. 4 if we suppose there is an increment in every step. We use the parameters shown in Table 2. In around 10 updates, the

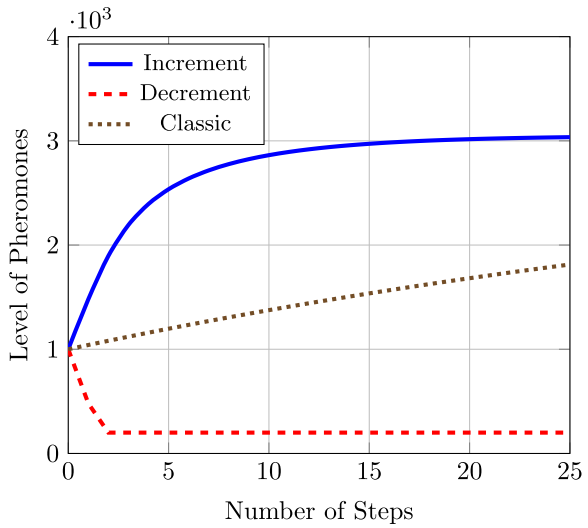

Fig. 6 Graph representing the evolution of pheromones

Table 2 Parameters of Morwilog

Name	Description	Simulation
T_{max}	Maximum search time	62 s
$\tau[0]$	Initial number of pheromones	1000
ρ	Evaporation rate	0.02
w	Spreading of $\Delta\tau^{+,-}$ function	1000
$\Delta\tau_0^+$	Change of pheromones when $\tau[n] = \tau[0]$	500
τ_{min}	Minimum level of pheromones	100
τ_{min}^{atk}	Minimum level of ph. for attack	200
P_{tree}	Prob. of creating a new tree if no match	0.5
P_{jump}	Prob. of adding a random sequence	0.1

level of pheromone is very close to the upper limit, which in this case is 3051.

We want to compare the evolution of pheromones with the one in classic ACO, where the value of increment is fixed. To do that, we have also represented it in the curve "Classic." We have chosen an increment value of 61.02, the exact value for reaching the same upper limit as in the curve "Increment," 3015. We see that our approach reaches the limit much earlier than the classical one.

Following the same logic for $\Delta\tau^-$, we also arrive at a lower limit but negative this time. As we want the level of pheromones to be always positive, it is necessary to artificially set a minimum value τ_{min} so we force the quantity of pheromones to never go below it. The result of a succession of decrements in an arc starting from $\tau[0]$ is also represented in Fig. 6, in the curve "Decrement."

7.4 The algorithm

We called E_{clas} to the set of events coming from the Classifier. From this set, we only send to Morwilog suspicious events that could potentially be part of an attack, according to the verdict issued by the Orchestrator from the results returned by the Classifier. This data is divided in batches before going into Morwilog according to the maximum search time T_{max}, defined as a parameter. We define $E_{in}^M = (e_b, e_c, \ldots, e_a, \ldots)$ as a subset of suspicious events contained in E_{clas}, with its elements ordered in time ($a > c > b$) and with the time difference between two consecutive events in the list less than T_{max}. A better accuracy in the detection of suspicious events by the Classifier allows incrementing T_{max} preserving at the same time the performance of the system, as Morwilog has to process a lower number of events. With a higher T_{max}, the system is able to detect multi-step attacks lasting longer.

It is important to note that we use a slightly different notation in OMMA than the one presented in the original Morwilog paper [9].

The sequence of actions performed by each *morwi* is summarized in pseudocode form in Algorithm 1. We now explain the whole process carried out by the *morwi*, which begins when a suspicious event e_i is chosen by the Orchestrator and arrives at Morwilog.

Algorithm 1 Morwi algorithm

Require: $E_{in}^M \subseteq E_{clas}$; $e_i \in E_{in}^M$
Ensure: $s \in S_{E_{in}^M}$, $\delta_j \in \Delta$; *isresult* $\in$ {true, false}

isresult $\leftarrow$ false
$\delta_j \leftarrow \delta \in \Delta \mid e_i \odot \kappa_0 = 1$
if $\delta_j = \varnothing$ **then**
  if random(0,1) $< P_{tree}$ **then**
    $s \leftarrow$ random sequence $\in S_{E_{in}^M} \mid s_0 = e_i$
    $\delta_j \leftarrow$ create_tree(s)
    $\tau_l \leftarrow \tau[0]$
  end if
else
  Add e_i to s
  $\kappa_n \leftarrow \kappa_0$ in δ_j
  while F_n in $\kappa_n \neq \varnothing$ **do**
    $C, s' \leftarrow$ find_nodes(E_{in}^M, i, F_n)
    if $C \neq \varnothing$ **then**
      $(\kappa_m, e_i, \tau_l) \leftarrow c \in C$, rand τ-based choice
      if $e_i = s'_0$ **then**
        Add branch to κ_n from s', $\tau \leftarrow \tau_{min}^{atk}$
        Append s' to s
        isresult $\leftarrow$ true
        return *isresult*
      else
        $\kappa_n \leftarrow \kappa_m$
        Add e_i to s
      end if
    else
      return *isresult*
    end if
  end while
end if
if $\tau_l \geq \tau_{min}^{atk}$ **then**
  isresult $\leftarrow$ true
end if
return *isresult*

First of all, the *morwi* checks if there is already Δ in a tree δ_j with an upper node κ_0 such that $e_i \odot \kappa_0 = 1$. If there is no tree matching the event ($\delta_j = \varnothing$), with a certain probability P_{tree}, the *morwi* forms a sequence from random events in E_{in}^M. This sequence has e_i as its first element.

The length of the sequence can be statically defined in the system or be a random variable, which could depend on the average number of actions for reaching a critical piece of the network. The sequence $s = (s_0, s_1, \ldots, s_{L_s}) \in S_{E_{in}^M}$ is considered as a result by the *morwi*, which invokes

the function *create_tree* for creating a tree with a number of nodes equal to the number of events in the sequence, every one of them being traversed by the same path. The number of pheromones assigned to the links between nodes is $\tau[0]$. In Eq. 5, we can find the formal definition of the tree generated by *create_tree*, knowing that $s_p \odot e_p^* = 1, s_p \odot \kappa_p = 1 \; \forall s_p \in s$.

$$\delta_j = \{\kappa_p\} = \begin{cases} \left(e_0^*, (\kappa_1, \tau[0]), \varnothing\right) & p = 0 \\ \left(e_p^*, (\kappa_{p+1}, \tau[0]), \kappa_{p-1}\right) & 0 < p < L_s \\ \left(e_{L_s}^*, \varnothing, \kappa_{L_s-1}\right) & p = L_s \end{cases} \tag{5}$$

If a tree δ_j is found, the *morwi* starts going through the tree starting in node κ_0. This process is repeated node after node until a node without children ($F_n = \varnothing$) is found or a random sequence is chosen. The reader can find the method *find_nodes* described in Algorithm 2.

Before continuing, we have to define the vector C, which contains a set of vectors. Each of the vectors contains an event e_a, a node κ_m matching to it, and the level of pheromones $\tau_{x,m}$ from the arc leading to κ_m in the tree.

The method *find_nodes* looks for events matching any of the children of κ_n among events coming after e_i in E_{in}^M, storing them in an instance of C together with the node $\kappa_m \in F_n$ representing it and $\tau_{n,m}$ associated to it. Apart from that, with a certain probability P_{jump}, it chooses a random sequence s' and adds its first event to C with τ_{min}^{atk} as the level of pheromones.

Algorithm 2 find_nodes algorithm

Require: $E_{in}^M \subseteq E_{clas}$; i, index of e_i; F_n, set of pairs $\mathbb{K}/\mathbb{R}$
Ensure: $C = \left\{(\kappa_m, e_a, \tau_{x,m}) \in \left(\mathbb{K}, E_{in}^M, \mathbb{R}\right)\right\}$; $s' \in S_{E_{in}^M}$

for each $\kappa_m \in F_n$ **do**
  During time $< T_{max}$, find $e_q \mid q > i, e_q \odot \kappa_m = 1$
  Add $(\kappa_m, e_q, \tau_{n,m})$ to C if e_q found
end for
if random(0,1) $< P_{jump}$ **then**
  $s' \leftarrow$ random $\in S_{E_{in}^M} \mid n > i \; \forall s_n \in s'$
  Add $\left(\{e_p^*, \varnothing, \varnothing\}, s'_0, \tau_{min}^{atk}\right)$ to $C \mid s'_0 \odot e_p^* = 1$
else
  $s' \leftarrow \varnothing$
end if

Once the *morwi* has the vector of found events and their corresponding nodes, it chooses the next event to jump from this list. Following a weighted random selection, the event with higher level of pheromones has more probabilities to be chosen. If the event starting the random sequence s' is chosen, then the sequence should be added as a new branch to tree δ_j with pheromones τ_{min}^{atk} in each node-to-node link. In this case, this is the solution proposed by the *morwi*.

7.5 Human feedback

We have called *Morwihill* the module that generates the *morwis* and manages the results returned by them. We can find the pseudocode describing the functioning of this module in Algorithm 3. This module creates the *morwi* and receives the returned result $s \in S_{E_{in}^M}$ once the execution is finished. Then, it can obtain the sequence of nodes that the *morwi* has traversed in the construction of s. If it is a valid result, it evaporates the pheromones leading to the nodes with the same ancestor of the last node in the sequence, according to Eq. 2.

Algorithm 3 Morwihill algorithm

Require: $E_{in}^M \subseteq E_{clas}$
for each e_i in E_{in}^M **do**
  $s, \delta_j, isresult \leftarrow$ Morwi$\left(E_{in}^M, e_i\right)$
  if $isresult =$ true **then**
    $(\kappa_0, \ldots, \kappa_l)$ path from δ_j matching s
    Evap. τ in paths, $\forall \kappa_n \in \delta_j$ with $\kappa_n^{(\alpha)} = \kappa_l^{(\alpha)}$
    if $\Pi(s) = 1$ **then** ▷ It is an attack
      Increment τ in link leading to κ_l
    else ▷ It is not an attack
      Decrement τ in link leading to κ_l
    end if
    Propagate change in τ from κ_l to κ_0
  end if
end for

After evaporation, the human expert determines if s is malicious or not. Feedback information is provided through the Visualizer module and makes the Morwilog algorithm able to learn. So far, we have considered binary feedback information. The human has to decide if it is an attack or if it is a false positive. For doing so, sequences returned by Morwilog are shown in a console. Each sequence contains all the information about the involved events for easing the investigation launched by the human expert.

In our case, the behavior of the human expert is modeled by the function $\Pi(s)$, which returns 1 if s is an attack and 0 otherwise. The level of pheromones leading to the last node is incremented if $\Pi(s) = 1$ and decremented if $\Pi(s) = 0$, following Eqs. 1 and 3, so attack sequences are reinforced and innocuous ones are penalized. Finally, changes in pheromones are propagated to the rest of the tree and the analysis thread is closed.

7.6 Example

We present here an example of how Morwilog works for better understanding the algorithm. The example

is inspired by recent WannaCry malware campaign[5]. The ransomware used in WannaCry attack tries to connect to an external domain name. The propagation mechanism inside the local network, which uses protocol SMB[6], starts when this connection is not successful. The infected machine scans the network looking for endpoints accepting connections in port TCP 445, the one used in SMB. If the endpoint uses SMBv1, a vulnerability[7] is exploited and the endpoint is also infected.

Figure 7 illustrates the example, and the kind of event matched by each node is shown in Table 1.

The steps followed by a *morwi* are described hereunder. Parameters used in the example are shown in Table 2.

(a) Each time a suspicious event e_i arrives at Morwilog, a *morwi* is generated and it looks for the tree whose root node matches the event, in this case δ_j. If it had not found any, it would have created with probability P_{tree} a new tree with one branch from a random sequence of actions.

(b) Starting in the root node, the *morwi* searches the events matching each of the nodes in the second level. The maximum search time is T_{max}. In this case, both events e_j and e_k corresponding to κ_1 and κ_2, respectively, are found. Now, the *morwi* has to decide a node for continuing its trip through the tree doing a pheromone-based weighted random selection. That means the path with a higher level of pheromones has a higher probability of being chosen. There is also a probability P_{jump} of choosing a random event instead of the ones proposed in the tree. This allows the exploration of other events.

(c) The most probable option is κ_1. Imagine the *morwi* chooses e_j, which corresponds to node κ_1. Now it has to search the events matching the nodes in the following level. In this case, no event is found corresponding to κ_4, which represents the exploitation of a vulnerability in SMBv2[8]. Only events e_l and e_n are matching κ_3 and κ_5. The *morwi* needs to choose between these two nodes, the first one having a higher probability.

(d) We can imagine the *morwi* chooses κ_3. It stops because it has arrived at the end of the tree. Finally, if we follow the most probable paths, the sequence (e_i, e_j, e_l) is returned as a possible attack since the level of pheromones for the last arc is higher than τ^{atk}_{min}, which in this case is 200. The resulting sequence is the one followed by WannaCry malware for propagating the infection.

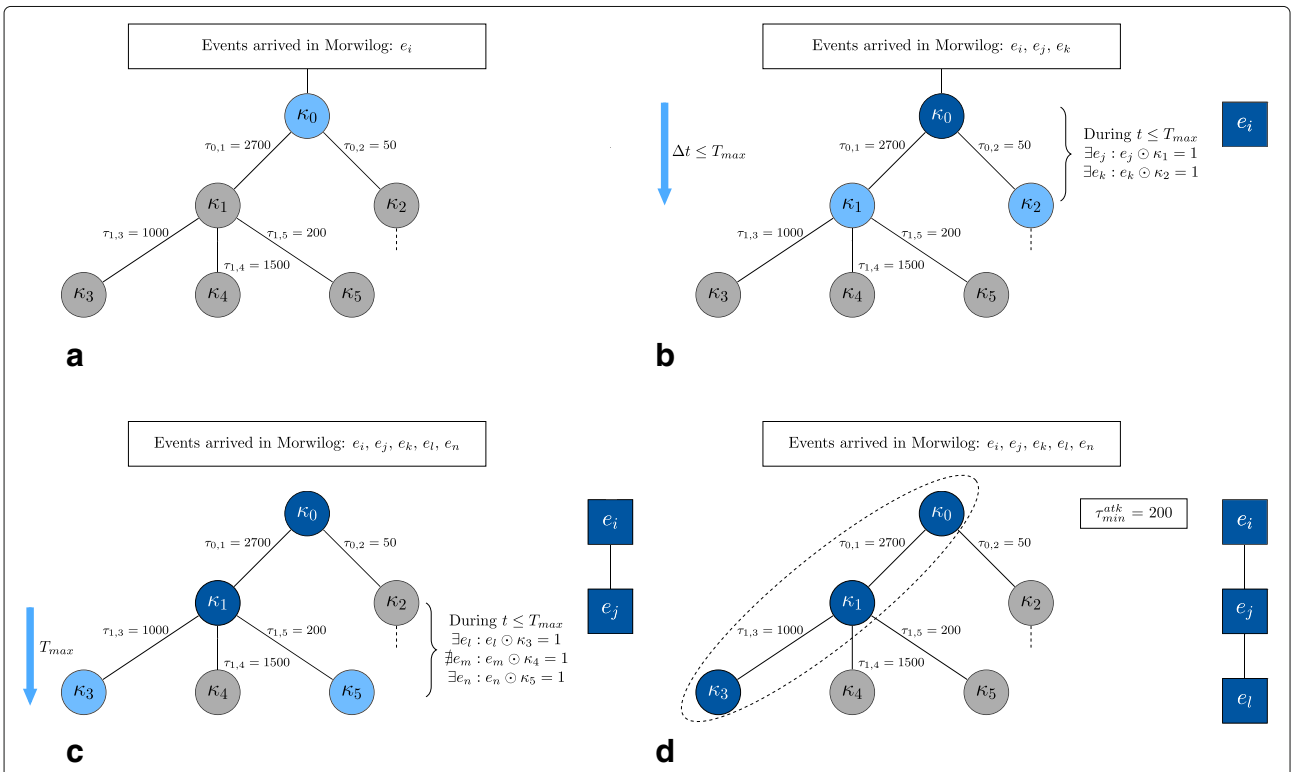

Fig. 7 Example about the progress of a *morwi* through an event tree with parameters in Table 2. **a** e_i arrives, *morwi* chooses tree δ_j with $e_i \odot \kappa_0 = 1$, **b** *morwi* has to choose between found events in level 2, **c** Pheromone-based weighted random selection, **d** (e^*_0, e^*_1, e^*_3) and (e_i, e_j, e_l) as possible attack if $\tau_{1,3} > \tau^{atk}_{min}$

The following step is to update the pheromones in the tree according to the feedback from the human expert. As we know, there are two processes of pheromone change, the evaporation and the increment/decrement. Evaporation avoids stagnation, and it is proportional to the previous pheromone level by the evaporation rate. It is applied to all the sibling nodes of the last node of the selected sequence. The increment/decrement is a way of rewarding good paths and punishing bad ones, and it has an exponential dependence on the pheromone level.

The next step is to evaporate pheromones in the last arc of selected sequence and in the ones leading to its siblings. If the sequence is confirmed to be an attack, we increment the level of pheromones of the last arc of the sequence. Then, the changes are propagated up through the tree. We can see a schema presenting this process in Fig. 8.

In the WannaCry attack, it can happen that command "transaction2_secondary" in SMBv1, represented in node κ_3, is used for testing purposes and not with a malicious objective. In this remote case, the expert considers it is a false alert and the level of pheromones is decremented. Remember we have fixed a lower limit τ_{min} for avoiding negative values. The evolution of pheromones for a false positive is shown in Fig. 9.

7.7 The integration in OMMA

Morwilog is directly integrated in the Orchestrator of OMMA. The structure of this module after the integration of Morwilog is shown in Fig. 10. Morwilog closes the feedback loop for the modification of event trees. As we have described, it is composed of the central *Morwihill* and the *morwis* it generates. Its inputs are events marked as suspicious once they are processed by the Classifier. The Orchestrator itself, after the return obtained from the Classifier, chooses which events are potential members of a multi-step attack and sends them to this submodule. Feedback information is used to update the pheromones in the event trees. It is also sent to the Classifier, so the methods within it can also improve their results, changing their parameters or telling the Advisor to change the event trees stored in it.

8 Results

Among all the elements composing OMMA, we have already developed Morwilog as a submodule of the Orchestrator, which is explained in Section 7. Even if we have not yet completed the development of the Collector, we have also implemented some of its functionalities for being able to collect the events to be used by Morwilog. The system has been developed in C++.

We have performed a set of experiments for evaluating the performance of Morwilog. As we do not have the previous phases in OMMA yet, we can make Morwilog work with events already processed, as if they had been classified. The goal of these experiments is to show the positive effect of pheromone evolution. As we do not have a set of pre-built event trees yet, trees are built during the execution of the algorithm, using the mechanism used when no matching tree is found.

As we explained in Section 4, we have worked with an artificial dataset. We have created it using Splunk Event Generator[9]. It is composed of a total of 1038 different types of events, represented in the shape of logs. They are stored following the standard audit trail format developed by Bishop [65]. Event types are randomly taken by the generator program in different proportions according to the event type, and a random IP source is included. We have defined several devices and the probabilities of occurrence of an event from this device. Inside each one of them, we have also defined a probability of appearance of each

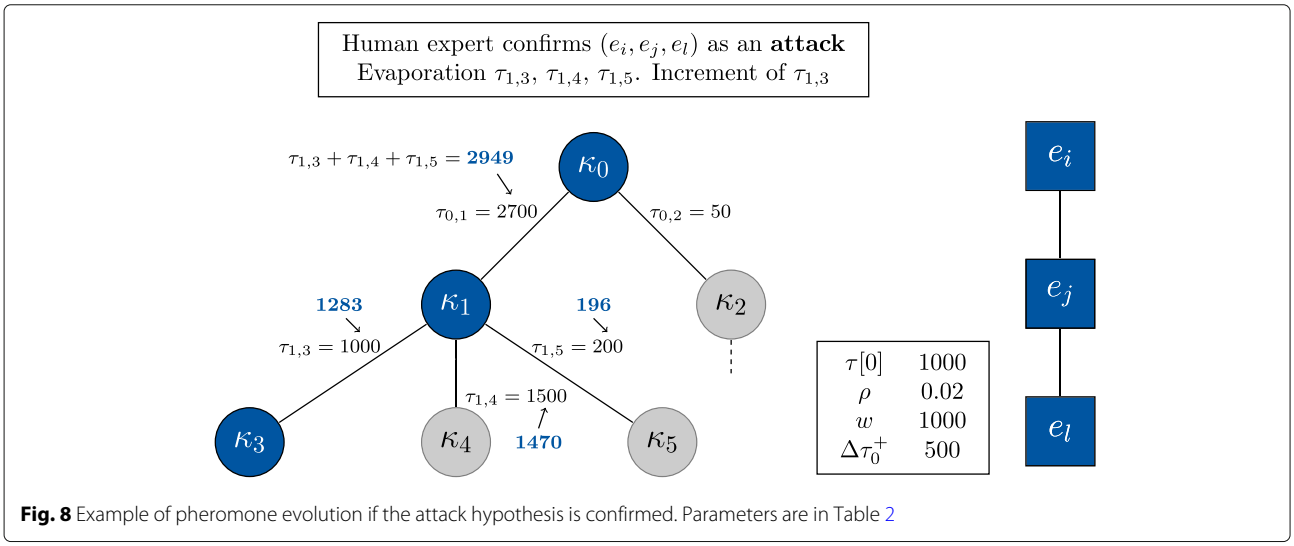

Fig. 8 Example of pheromone evolution if the attack hypothesis is confirmed. Parameters are in Table 2

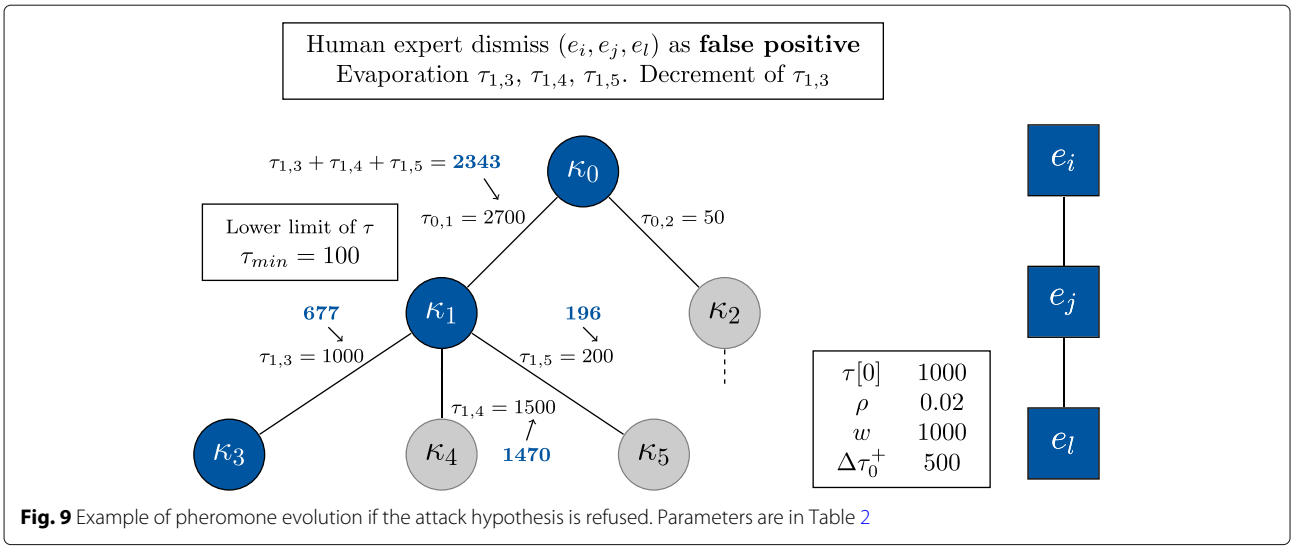

Fig. 9 Example of pheromone evolution if the attack hypothesis is refused. Parameters are in Table 2

possible type of event. Timestamps are generated by a counter, with intervals of increment that are also random.

Once the events have been generated, we inject sequences of events with the IP source as the overarching element. These sequences represent 40% of events in the dataset. Half of them are labelled as attacks, while the others are innocuous. In each dataset, there are at least 10 types of attacks and 40 types of innocuous sequences. The types have been randomly defined. The exact composition of an attack during the test does not matter, the important thing is that we are able to identify it as a different sequence. The length of the sequences is a parameter of each simulation.

Simulations have been carried out on an Intel Core i5 machine running at 1.4 GHz with 8 GB RAM. Each simulation has been repeated 50 times and then taken the

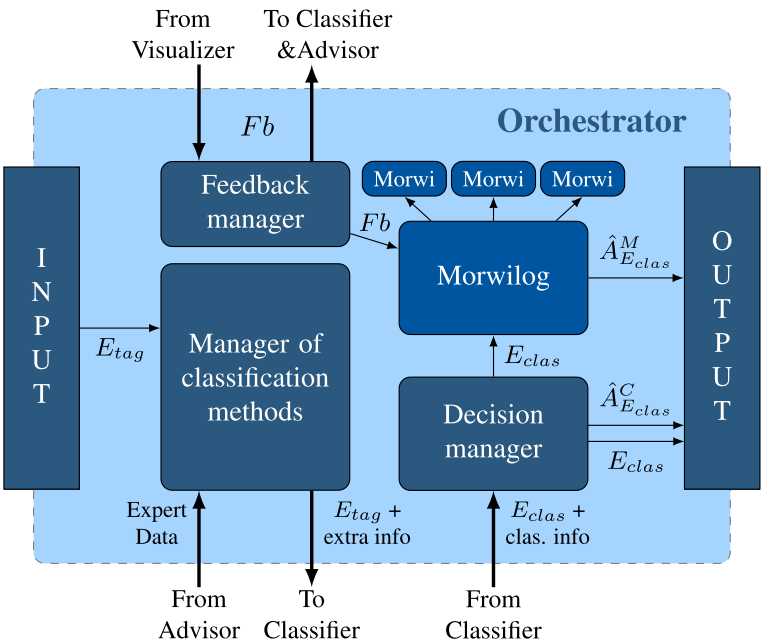

Fig. 10 Diagram of the structure inside the Orchestrator. Input is coming from the Matcher. Output goes towards the Logger, the Visualizer, and the Actuator

average of the results. Morwilog needs nine parameters to be determined before execution. The values chosen for these simulations are shown in Table 2.

The execution time grows linearly with the number of events to analyze, as we can see in Fig. 11.

The metrics for evaluating Morwilog have been taken from the classical definitions collected by Marchette [66]. The probability of detection (PD) and the probability of false alarm (PFA) are used. PD is the number of detections made correctly over the total number of attacks presented in the dataset. PFA is the quantity of false alerts over the total number of alerts generated by Morwilog.

Simulation results are shown in Fig. 12 as a ROC curve. Horizontal axis represents the PFA and vertical one, PD. Each point in the graph corresponds to the average result from 50 simulations over datasets with different sizes. Each one is tagged with the number of events used, from 100 to 40,000 (40k). We have represented the results for different lengths (L) of sequences.

On the other hand, we have also represented a ROC curve for simulations with the same parameters but removed the pheromone update. The results are shown in Fig. 13.

We have clearly better results when there are more events contained in the dataset. We see that the curves in Fig. 12 move closer to the upper left corner of the plot when the number of events analyzed are higher. There is also a variation in results with the length of the sequences. If they are higher, results are worse, because the sequences are more intercalated with normal traffic. Moreover, if we compare with the ROC curve in Fig. 13, we can verify the positive effect of pheromone update, since the results

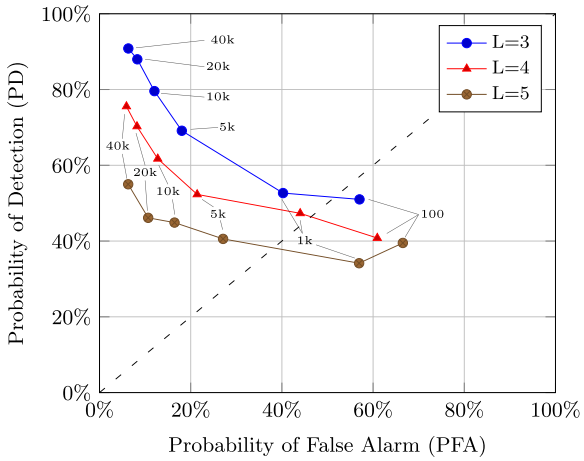

Fig. 12 ROC curve (PFA vs. PD) representing the results from simulations of Morwilog. Each point is the average of the result of 50 simulations with the parameters in Table 2. The size of the dataset is indicated next to the points. L is the length of injected sequences

obtained are much worse for any length of sequence in the pheronome-free case.

9 Discussion

Nowadays, organizations seem more protected than ever against cyberattacks. Systems, software, and protocols are created with security in mind from the first stages, and security devices contain more detection tools and mechanisms than ever. Infecting an organization seems harder than in the era when simple malware campaigns were successful. Consequently, attacks are ever more sophisticated. As they require a lot of resources, attackers give priority to large victims. The actions of the attackers are hidden

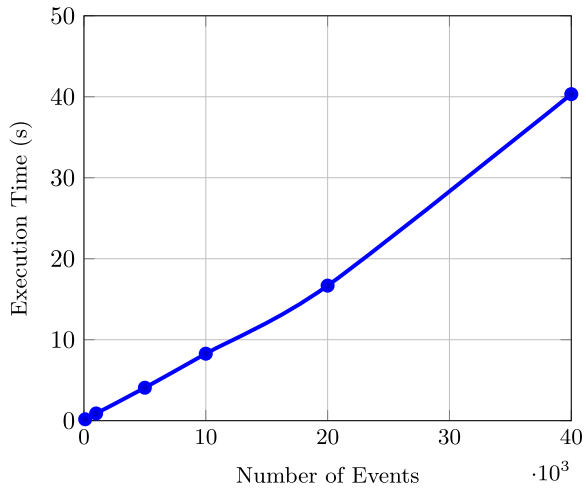

Fig. 11 Morwilog execution time vs. number of events in E_{in}. Each point is the average of the result of 50 simulations with the parameters in Table 2

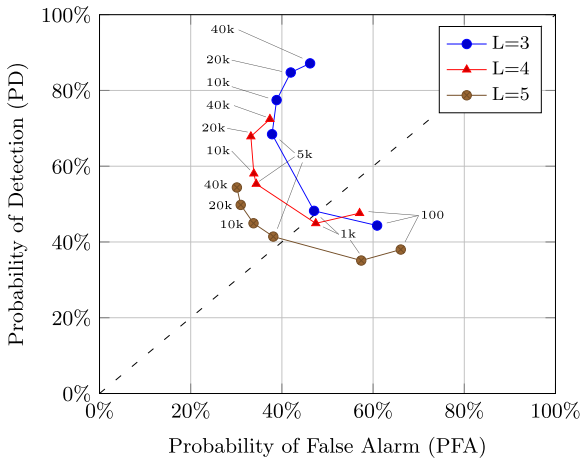

Fig. 13 ROC curve (PFA vs. PD) representing the results from simulations of Morwilog without doing the pheromone update. Conditions are the same as in Fig. 12

into the vast volume of events generated daily within the organization.

OMMA, the Engineering System we have presented in this paper, provides security researchers with a framework to develop effective algorithms for multi-step attack detection. It proposes a collaborative and open architecture where the feedback from human experience is a key piece. The full implementation of the system still has a long way to go, but first results are promising and provide a significant milestone in the definition of an open common architecture for alert correlation in the case of multi-step attacks. Other researchers can take advantage of the proposed solution and contribution to its future evolution.

9.1 Towards an open architecture

As stated in Section 4, one of our goals is to make OMMA an open system. Most of the research contributions to attack detection are controlled by private companies, who do not make public the details about the methods they use for keeping their competitive advantage. Finding multi-step threats is becoming an important issue for everybody. Only the collaboration between the public and private sectors and the open access to information about attacks and defenses can lead to the development of robust and performing systems for protecting society against current cyberattacks. Vendors should know about new exploits right after they are discovered, so they can develop patches for their products as soon as possible. This way, the effect of massive attacks making use of old undisclosed exploits could be mitigated. It is the case of the already cited WannaCry attack, which has the EternalBlue exploit, which has been disclosed two months before the attack, as one of its elements.

With OMMA, we propose a platform where detection methods with diverse origins can be adapted for working together in the detection of multi-step threats. Its modular design allows even the combination of different hardware as soon as a standard for communication between modules is defined.

9.2 External human feedback

We consider the human expert as a central piece in the detection process, as Legrand et al. [3] do. We take up their idea about the human experts possessing an acquired knowledge which is difficult to replicate in a machine as a simple set of rules. We will not be able to get rid of the human presence and develop a fully automated system. The expert has to be involved in the detection process for providing this know-how to the detection system, which can thereby improve its results.

Apart from this idea, which we firmly believe in after many contacts with security experts, we think another important reason why the security expert is necessary is that it is an actor external to the system. That means this person is able to check the consequences of a suspicious scenario using elements outside the network domain, while devices are restricted to it. For example, a network device cannot interrogate the personnel or remove infected disks from a server to do forensic investigation. Moreover, human creative thinking is still far from being reached by a machine, but its results can be introduced on it through a feedback module.

This explains the projection of OMMA on the feedback loop, which is used by Morwilog, the first element we have developed and tested. While in classical correlation manual intervention is required for the development of the rules, in our system it is incorporated during the analysis, as a continuous task. We expect our work can move researchers to consider the presence of security experts' role in detection, not only during preliminary rule definition and subsequent investigation phases.

9.3 Answer to challenges

We conclude the discussion by listing how OMMA offers a platform to give answer to the challenges in classic correlation identified in Section 2.

- *Rules are manually defined.* The proposed system is mainly oriented to the application of machine learning algorithms. These algorithms do not need the explicit definition of rules. They can automatically learn from verdict of the human expert and interpretation of past events. The Matcher module, which is based on rules, is a complement for detecting known attacks and alleviating the effort done by the other modules, but it is not the core of the system.
- *Detection accuracy depends on the expert's ability.* An error in a manually defined detection rule stays in the system until an expert finds and corrects it. However, an error in the verification of threats in OMMA is autocorrected by statistics. Punctual errors are statistically negligible if the proportion of correct answers is higher.
- *A lot of time is spent in log analysis.* OMMA is conceived for highlighting relevant events in terms of threat. The expert only needs to review the suspicious scenarios returned by the system. Diving into raw logs is not necessary either, as in the presented architecture every log arriving at the system is parsed, normalized, and enriched.
- *Rules are just suited to known threats.* In contrast, machine learning algorithms included in OMMA could detect unknown ones, especially unsupervised ones. Unknown attacks are never completely new, so some parts of it can be identified from historic data and from the feedback the human expert provides. However, trying to predict future attacks and

preserving the common parts in rule format is a titanic task.

- *Systems are difficult to maintain and control due to log heterogeneity*. As the system is not based on rules, we do not need to create new ones if unknown types of logs arrive to the system. Parsers need to be updated and automatic parsing methods need to be verified for the new events. Once this is done, the analysis methods use normalized events, so they can continue working without changes.

10 Conclusions

In this paper, we have presented OMMA, an open engineering system for multi-step attack detection with the assistance of a human security expert. The architecture of the system has been explained, and we have shown the details of Morwilog as a submodule integrated in OMMA. We have seen that the modular design of OMMA allows the adaptation to any network. The contribution of other researchers can be incorporated to any of the modules. As modules are independent, each contributor can work on their development without considering what is done in the others. So far, Morwilog, contained in the Orchestrator, and part of the Collector have been developed. Results from experiments on a representative artificial set of events have been presented.

Even if the results obtained from Morwilog show the positive effect of pheromone evolution, we still need to try on a heterogeneous labelled set of events with multi-step attacks on it. This brings us to the problem of finding proper open datasets in security research. OMMA is conceived under the perspective of open collaboration. Our aim is to establish the foundations of a platform for multi-step attack detection through event analysis where methods with different origins can work together. This framework already considers the inclusion of the human expert in the detection process as a key piece whose creative thinking we find indispensable for threat verification.

Our objective is to continue the development both of OMMA general framework and the instantiation of their elements. The next step is to formally define the interfaces of the different modules to promote standardization. The detection of traces left by multi-step attacks in log files and the generation of event trees are key challenges for enabling production-level deployment of the OMMA framework.

Endnotes

[1] https://www.enisa.europa.eu/publications/info-notes/wannacry-ransomware-outburst

[2] https://www.fireeye.com/blog/threat-research/2012/05/advanced-persistent-threat-apt-malware.html

[3] https://www.alienvault.com/products/ossim

[4] http://www.mlsecproject.org/blog/the-importance-of-good-labels-in-security-datasets

[5] https://www.us-cert.gov/ncas/alerts/TA17-132A

[6] https://logrhythm.com/de/blog/a-technical-analysis-of-wannacry-ransomware/

[7] https://logrhythm.com/de/blog/using-netmon-to-detect-wannacry-initial-exploit-traffic/

[8] http://seclists.org/fulldisclosure/2009/Sep/39

[9] https://github.com/splunk/eventgen

Acknowledgements
We thank all colleagues from the HuMa project for their valuable feedback on the Morwilog model, as well as the CSTB Team of the ICube laboratory for their support.

Funding
This work was partially supported by the French Banque Publique d'Investissement (BPI) under program FUI-AAP-19 in the frame of the HuMa project, as well as by the ICube SENSAI Project.

Availability of data and materials
Data and material cannot be shared because industrial contributors to the HuMa project reserve the right to not publish code, details about the implementation, or datasets until at least 6 months after the final presentation of the project.

Authors' contributions
JN was responsible for the concretion of the architecture, development and conception of Morwilog, main tasks of redaction and exposition, composition of images, and execution of experiments. VL contributed in the redaction and exposition, conception of the main ideas, and revision and corrections. AD participated in the revision and corrections, and additional precisions. PP contributed in the redaction and exposition, revision and corrections, collaboration in the conception of Morwilog, concretion of the architecture, and definition of the structure of the article. All authors read and approved the final manuscript.

Authors' information
Julio Navarro is a PhD student at the "Complex Systems and Translational Biology" (CSTB), a research team part of the ICube Laboratory at the University of Strasbourg. His research topic is the development of multi-step attack detection methods from network traces. He obtained his M.S. in Telecommunication Engineering from the University of Granada in 2012 with the highest results. He spent his last year abroad at the University of California Los Angeles (UCLA), where he presented his Master Project on mobile communications. He got the Spanish National Degree Award to the second most outstanding graduate in Engineering. Before starting his PhD, he spent three years in Madrid working as a Security Project Engineer.
Véronique Legrand is a Professor in Computer Security at Cnam (Conservatoire national des arts et métiers) and Project Manager of Innovation and Research at Intrinsec. She is currently one of the coordinators of the HuMa project. Before 2016, she held a professor position at INSA Lyon for 15 years. She got her PhD in the domain of Computer Security in 2013.
Aline Deruyver is a tenured senior Associate Professor in computer science at the University of Strasbourg. She is a member of the CSTB (Complex System and Translational Bioinformatic) team from the ICube Laboratory of Strasbourg, France. She obtained a PhD in Computer Science from the University of Lille I in 1991, and she obtained an accreditation to supervise research from the University of Auvergne in 1999. She is a member of the Technical Committee TC15 of IAPR: representation of images by graphs and a proofreader in the workshop of the technical committee "Graph Based Representation For Pattern Recognition".
Dr. Pierre Parrend is the head of the Computer Science and Mathematics Department at ECAM Strasbourg-Europe engineer school. He is a member of the research team "Complex Systems and Translational Biology" at ICube Laboratory of the University of Strasbourg and the head of the e-laboratory

"4PFactory: the factory of the future" of the UNESCO Unitwin Complex System-Digital Campus. His research interests encompass attack detection, software security, artificial immune ecosystems, evolutionary strategies for optimization and anomaly detection, and emergent properties of human organizations.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Laboratoire ICube, Université de Strasbourg, 11, Rue Humann, Strasbourg, France. [2]CEDRIC, Conservatoire National des Arts et Métiers (CNAM), 2 Rue Conté, Paris, France. [3]Intrinsec Security, 215 Avenue Georges Clemenceau, Nanterre, France. [4]Unitwin UNESCO Complex System-Digital Campus, Paris, France. [5]ECAM Strasbourg-Europe, 2, Rue de Madrid, Schiltigheim, France.

Received: 30 November 2017 Accepted: 2 April 2018
Published online: 02 May 2018

References

1. M-Trends 2017: a view from the front lines [Generic]. FireEye (2017). The publication date is March 14, 2017. https://www.fireeye.com/blog/threat-research/2017/03/m-trends-2017.html. Accessed 23 Apr 2018
2. D Jaeger, M Ussath, F Cheng, C Meinel, in *IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud)*. Multi-step attack pattern detection on normalized event logs (IEEE, New York, 2015), pp. 390–398
3. V Legrand, P Parrend, P Collet, S Frénot, M Minier, in *Cesar 2014: Detection et reaction face aux attaques informatiques*. Vers une architecture «big-data» bio-inspirée pour la détection d'anomalie des SIEM (Rennes, France, 2014)
4. E Crawley, O De Weck, C Magee, J Moses, W Seering, J Schindall, et al., *The influence of architecture in engineering systems*. (MIT, Cambridge, 2004)
5. M Vogel, S Schmerl, in *OASIcs-OpenAccess Series in Informatics*. Efficient distributed intrusion detection applying multi step signatures, vol. 17 (Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Wadern, 2011), pp. 188–193
6. R Abreu, D Bobrow, H Eldardiry, A Feldman, J Hanley, T Honda, et al., in *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015)*. Diagnosing advanced persistent threats: a position paper, (2015), pp. 193–200
7. J Navarro, V Legrand, S Lagraa, J François, A Lahmadi, G De Santis, et al., in *The 10th International Symposium on Foundations & Practice of Security (FPS)*. HuMa: a multi-layer framework for threat analysis in a heterogeneous log environment (Springer International Publishing, Nancy, 2017)
8. Aristophanes, *Clouds. Wasps. Peace. Loeb Classical Library*. (Hardvard University Press, Cambridge, MA, 1998)
9. J Navarro, A Deruyver, P Parrend, in *IEEE Symposium Series on Computational Intelligence (SSCI)*. Morwilog: an ACO-based system for outlining multi-step attacks (IEEE, Athens, 2016)
10. Standard on logging and monitoring [Standard]. European Commission (2010). https://www.eba.europa.eu/documents/10180/1449046/Annex+5+Standard+on+Logging+and+Monitoring.pdf/4e9f17de-4589-424c-a670-c0cdc1b5f67b. Accessed 23 Apr 2018
11. J Ya, T Liu, H Zhang, J Shi, L Guo, in *IEEE Military Communications Conference (MILCOM)*. An automatic approach to extract the formats of network and security log messages (IEEE, Tampa, 2015), pp. 1542–1547
12. D Jaeger, A Azodi, F Cheng, C Meinel, in *IFIP International Conference on Information Security Theory and Practice*. Normalizing security events with a hierarchical knowledge base (Springer, Heraklion, 2015), pp. 237–248
13. I Friedberg, F Skopik, G Settanni, R Fiedler, Combating advanced persistent threats: from network event correlation to incident detection [Journal Article]. Comput. Secur. **48**, 35–57 (2015)
14. J Navarro, ¿Quién teme a la APT feroz? [Magazine Article]. eSecurity. **50**, 52–57 (2014)
15. G Suarez-Tangil, E Palomar, JM De Fuentes, J Blasco, A Ribagorda, in *Proceedings of the 2nd International Workshop on Computational Intelligence in Security for Information Systems (CISIS'09)*. Automatic rule generation based on genetic programming for event correlation, vol. 63 (Springer, Burgos, 2009), pp. 127–134
16. M Hasan, B Sugla, R Viswanathan, in *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*. A conceptual framework for network management event correlation and filtering systems (IEEE, Boston, 1999), pp. 233–246
17. KM Kavanagh, O Rochford, *Magic quadrant for security information and event management [Generic]*. (Gartner, 2015)
18. A Müller, *Event correlation engine [Master's Thesis]*. (Eidgenössische Technische Hochschule Zürich, 2009)
19. TB Oliver Rochford, KM Kavanagh, *Critical capabilities for security information and event management*. (Gartner, Stamford, 2016)
20. KM Kavanagh, O Rochford, Magic quadrant for security information and 1847 event management [Generic]. AlienVault (2014). https://www.alienvault.com/doc-repo/USM-for-Government/all/Lifecycle-of-a-Log.pdf. Accessed 23 Apr 2018
21. F Alserhani, M Akhlaq, IU Awan, AJ Cullen, P Mirchandani, in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*. MARS: Multi-stage Attack Recognition System (IEEE, Perth, 2010), pp. 753–759
22. B Chen, J Lee, AS Wu, in *Fourth IEEE International Workshop on Information Assurance (IWIA'06)*. Active event correlation in Bro IDS to detect multi-stage attacks (IEEE, London, 2006), pp. 16–50
23. H Du, DF Liu, J Holsopple, SJ Yang, in *Proceedings of 19th International Conference on Computer Communications and Networks*. Toward ensemble characterization and projection of multistage cyber attacks (IEEE, Zurich, 2010), pp. 1–8
24. MY Huang, RJ Jasper, TM Wicks, A large scale distributed intrusion detection framework based on attack strategy analysis [Journal Article]. Comm. Com. Inf. SC. **31**(23), 2465–2475 (1999)
25. X Qin, W Lee, in *20th Annual Computer Security Applications Conference*. Attack plan recognition and prediction using causal networks (IEEE, Tucson, 2004), pp. 370–379
26. S Mathew, S Upadhyaya, in *IEEE Military Communications Conference (MILCOM)*. Attack scenario recognition through heterogeneous event stream analysis (IEEE, Boston, 2009), pp. 1–7
27. P Ning, Y Cui, DS Reeves, in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. Constructing attack scenarios through correlation of intrusion alerts (ACM, Washington DC, 2002), pp. 245–254
28. ST Eckmann, G Vigna, RA Kemmerer, STATL: an attack language for state-based intrusion detection [Journal Article]. J. Comput. Secur. **10**(1-2), 71–103 (2002)
29. M Meier. Intrusion Detection effektiv!: Modellierung und Analyse von Angriffsmustern (Springer-Verlag, Berlin, 2007)
30. M Ussath, F Cheng, C Meinel, in *IEEE Symposium Series on Computational Intelligence (SSCI)*. Automatic multi-step signature derivation from taint graphs (IEEE, Athens, 2016), pp. 1–8
31. M Vogel, S Schmerl, H König, in *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Efficient distributed signature analysis (Springer, Nancy, 2011), pp. 13–25
32. C Kruegel, T Toth, C Kerer, in *International Conference on Information Security and Cryptology (ICISC)*. Decentralized event correlation for intrusion detection [Journal Article] (Springer, Seoul, 2002), pp. 59–95
33. Z Anming, J Chunfu, in *Fifth World Congress on Intelligent Control and Automation (WCICA)*. Study on the applications of Hidden Markov Models to computer intrusion detection, vol. 5 (IEEE, Hangzhou, 2004), pp. 4352–4356
34. F Skopik, I Friedberg, R Fiedler, in *2014 IEEE Innovative Smart Grid Technologies Conference (ISGT)*. Dealing with advanced persistent threats in smart grid ICT networks (IEEE, Washington DC, 2014), pp. 1–5
35. P Giura, W Wang, in *Proceedings of the 2012 International Conference on Cyber Security*. A context-based detection framework for Advanced Persistent Threats (IEEE Computer Society, Washington DC, 2012), pp. 69–74
36. P Giura, W Wang, Using large scale distributed computing to unveil Advanced Persistent Threats [Journal Article]. Sci. J. **1**(3), 93–105 (2012)
37. K Pei, Z Gu, B Saltaformaggio, S Ma, F Wang, Z Zhang, et al., in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. HERCULE: attack story reconstruction via community discovery on correlated log graph (ACM, Los Angeles, 2016), pp. 583–595

38. J Navarro, A Deruyver, P Parrend, A systematic survey on multi-step attack detection. Comput. Secur. **76**, 214–249 (2018). https://doi.org/10.1016/j.cose.2018.03.001
39. M Dorigo, T Stützle, *Ant colony optimization*. (MIT Press, Cambridge, 2004)
40. G Theraulaz, E Bonabeau, A brief history of stigmergy [Journal Article]. Artif Life. **5**(2), 97–116 (1999)
41. DM Gordon, *Ant encounters: interaction networks and colony behavior*. (Princeton University Press, Princeton, 2010)
42. M Dorigo, V Maniezzo, A Colorni, *Positive feedback as a search strategy*. (Politecnico di Milano, Milan, 1991), pp. 91–016
43. M Dorigo, *Optimization, learning and natural algorithms [Ph.D. Thesis]*. (Politecnico di Milano, Italy, 1992)
44. S Haldenbilen, C Ozan, O Baskan, *An ant colony optimization algorithm for area traffic control*. (INTECH Open Access Publisher, London, 2013)
45. S Fernandez, S Alvarez, D Díaz, M Iglesias, B Ena, in *International Conference on Swarn Intelligence (ANTS 2014)*. Scheduling a galvanizing line by ant colony optimization (Springer, Brussels, 2014), pp. 146–157
46. G Valigiani, *Développement d'un paradigme d'optimisation par Hommilià et application á l'enseignement assistè par ordinateur sur Internet [Ph.D. Thesis]*. (Université du Littoral Côte d'Opale, Dunkerque, 2006)
47. G Valigiani, E Lutton, C Fonlupt, P Collet, Optimisation par "hommilière" de chemins pédagogiques pour un logiciel d'e-learning [Journal Article]. Tech. Sci. Inform. **26**(10), 1245–1267 (2007)
48. P Mahanti, M Al-Fayoumi, S Banerjee, Simulating targeted attacks using research honeypots based on ant colony metaphor [Journal Article]. Eur. J. Sci. Res. **17**(4), 509–522 (2005)
49. Z Zhang, PH Ho, Janus: a dual-purpose analytical model for understanding, characterizing and countermining multi-stage collusive attacks in enterprise networks [Journal Article]. J. Netw. Comput. Appl. **32**(3), 710–720 (2009)
50. GA Fink, JN Haack, AD McKinnon, EW Fulp, Defense on the move: ant-based cyber defense [Journal Article]. IEEE Secur. Priv. **12**(2), 36–43 (2014)
51. X Hui, W Min, Z Zhi-ming, in *International Conference on Industrial Mechatronics and Automation (ICIMA)*. Using Ant Colony Optimization to modeling the network vulnerability detection and restoration system (IEEE, Chengdu, 2009), pp. 21–23
52. M Kemiche, R Beghdad, in *Science and Information Conference (SAI)*. CAC-UA: a Communicating Ant for Clustering to Detect Unknown Attacks (IEEE, London, 2014), pp. 515–522
53. DP Jeyepalan, E Kirubakaran, Agent based parallelized intrusion detection system using Ant Colony Optimization [Journal Article]. Int. J. Comput. Appl. (IJCA). **105**(10), 1–6 (2014)
54. G Fernandes, LF Carvalho, JPC JRodrigues, ML Proença, Network anomaly detection using IP flows with principal component analysis and Ant Colony Optimization [Journal Article]. J. Netw. Comput. Appl. **64**, 1–11 (2016)
55. W Feng, Q Zhang, G Hu, JX Huang, Mining network data for intrusion detection through combining SVMs with Ant Colony Networks [Journal Article]. Futur. Gener. Comp. Sy. **37**, 127–140 (2014)
56. MS Abadeh, J Habibi, A hybridization of evolutionary fuzzy systems and Ant Colony Optimization for intrusion detection [Journal Article]. ISC Int. J. Inf. Secur. (ISeCure). **2**(1), 33–46 (2015)
57. MNK Abdurrazaq, BR Trilaksono, B Rahardjo, DIDS using cooperative agents based on ant colony clustering [Journal Article]. J. ICT Res. Appl. **8**(3), 213–233 (2015)
58. C Kolias, G Kambourakis, M Maragoudakis, Swarm intelligence in intrusion detection: a survey [Journal Article]. Comput. Secur. **30**(8), 625–642 (2011)
59. A Shostack, *Threat modeling: designing for security*. (Wiley, Hoboken, 2014)
60. F Guigou, P Parrend, P Collet, in *First Complex Systems Digital Campus World E-Conference*. An artificial immune ecosystem model for hybrid cloud supervision (Springer, Tempe, 2015), pp. 71–84
61. S Kobayashi, K Fukuda, H Esaki, in *Proceedings of The Ninth International Conference on Future Internet Technologies*. Towards an NLP-based log template generation algorithm for system log analysis (ACM, Tokyo, 2014), p. 11
62. R Gerhards, *The syslog protocol. RFC Editor; 2009. 5424*. Available from: https://tools.ietf.org/html/rfc5424. Accessed 23 Apr 2018
63. J Pokorny, Proto-Indo-European etymological dictionary. A Revised Edition of Julius Pokorny's Indogermanisches Etymologisches Wörterbuch. Indo-Eur. Lang. Revival Assoc. (2007). Available from: https://marciorenato.files.wordpress.com/2012/01/pokorny-julius-proto-indo-european-etymological-dictionary.pdf. Accessed 23 Apr 2018
64. J Clackson, *Indo-European linguistics: an introduction*. (Cambridge University Press, Cambridge, 2007)
65. M Bishop, in *Proceedings of the 18th National Information Systems Security Conference*. A standard audit trail format (DTIC Document, Baltimore, 1995), pp. 136–145
66. DJ Marchette, *Computer intrusion detection and network monitoring: a statistical viewpoint*. (Springer Science & Business Media, New York, 2001)