Morwilog: an ACO-based System for Outlining Multi-Step Attacks

Julio Navarro-Lara*†, Aline Deruyver*† and Pierre Parrend‡*†
*ICube Laboratory, Université de Strasbourg - 4 Rue Blaise Pascal, 67081 Strasbourg, France
Email: navarrolara@unistra.fr, aline.deruyver@unistra.fr, parrend@unistra.fr
†Complex System Digital Campus (UNESCO Unitwin) - http://cs-dc.org
‡ECAM Strasbourg-Europe - 2 Rue de Madrid, 67300 Schiltigheim, France

***Abstract*—Threat detection is one of the basic mechanisms for protecting a network, as prevention does not suffice. Finding an attack is difficult because the most harmful ones are specially prepared against a specific victim and crafted for the first time. The contribution of a human expert is still needed for their detection, no matter how effective automatic methods used nowadays can appear. Moreover, in many occasions intrusions can only be efficiently detected by analyzing its effects on more than one element in the network. Event and alert recollection offers a way to centralize information from a heterogeneous set of sources. Then, it can be normalized to a common language and analyzed as a whole by a security system. In this paper we propose Morwilog, an ant-inspired method for standing out the relationship between actions belonging to the same complex attack. Morwilog is conceived as a framework for alert correlation to be integrated in a multi-modular security system. Reinforcement learning is incorporated to it thanks to feedback from a human security expert.**

I. Introduction

Threat detection is acquiring a special relevance in network protection, even if prevention is still a key piece. This is required by current fast changing, complex and diverse IT environments, whose wide range of available services brings an uncontrollable amount of vulnerabilities. Organizations assume they will be attacked sooner or later and they search mechanisms to detect intrusions as early as possible and thereupon start the mitigation or remediation [1].

We can classify threats according to many criteria, as severity, kind of target or attacker's profile. But the differentiation between known and unknown threats is the one better addressing the main problem security detection faces, even if it is vague (as it is not addressed *who knows*) and not technical. We qualify a threat as *known* if it has been widely detected, analysed and understood by the security industry, so a signature characterising it has been developed for most of the intrusion detection platforms and it could be detected using pattern matching. For identifying the *unknown* ones, an automatic process has to be developed based on how attacks work and how they differ from normal traffic.

We can consider three different sources of information for facing threat detection: network traffic, endpoint operations and log files. Each source generates a particular type of data for which a security system has to consider a different perspective. In the network perspective, threats are detected thanks to the information extracted from traffic, whether through direct analysis on packets (e.g. pattern matching based on attack signatures) or flow summary examination. The endpoint orientation refers to the analysis of processes and programs in a server or a host, either real or simulated. The last category is the log or event analysis, based on the collection, aggregation and correlation of events registered by each device. We can think of it as an indirect way of working with the other two sources through abstract representations, as from log files we can extract both network and endpoint information.

According to the Standard on Logging and Monitoring published by the European Commission in 2010 [2], we can define an *event* as "an identifiable action that happens on a device and is recorded in a log entry". A *log entry*, also simply referred to as *log*, is an expression of an event using a language previously defined and dependent of the type of device that registers it [3]. Logs provide a valuable source of information for knowing what is happening in a network. This includes, of course, the detection of anomalous or threatening events or chains of events. 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 information with different origins can be merged for discovering threats that would be impossible to discover with information from a single source.

In this paper we present a new approach for revealing the causal relationship between alerts or events through reinforcement learning. Events collected by the system are filtered and generate artificial agents based on ant's behaviour. The framework in charge of that has been called Morwilog and it is intended to find the traces of multi-step attacks through the interconnection of events.

The rest of the paper is organized as follows. We first present the problems and challenges in Section II. In Section III, we review the related work. We explain the Morwilog system in Section IV and we present the experimental results in Section V. We finish the paper stating the conclusions and future work in Section VI.

II. Problems and Challenges

Event correlation [4] consists on the combination of an ensemble events into *composite* events [5], created as a conceptual construct as the event is not directly observable. Security systems recollecting logs and applying correlation

[6] are known as SIEM (System of Information and Event Management) in the industry. Their goal is not only detecting new incidents but also unifying and correlating alerts generated by other security devices. Log analysis seems to be the best approach for detecting attacks that cannot be characterised by less than two events, called multi-step attacks [7].

However, the heterogeneity of this set of information, joined to its great increment in terms of volume and variety in the last few years, hinders the analysis. Methods for event analysis has to be continuously reinvented and improved to be adapted to new circumstances. Automation in threat detection is a key objective, reducing the time security experts have to employ looking for incidents in the records.

The classical approach to event correlation in commercial platforms is *rule-based reasoning* [8], based on identifying the match between rules written by an expert and events in the set. There exists many ways for evaluating the rules, the simplest being the linear check of each rule against the incoming logs.

This method is used for example by OSSIM, whose open code has widely disseminated the idea of security event correlation. The event traverses several steps in the form of log [9], like assignation of risk score to the asset or comparison with reputation data, among others, before arriving to the correlation engine. This engine contains the rules defining the sequences of events representing a threat. The event is checked against the beginning of those rules. If matching conditions are met, an alarm is triggered and a separate process starts in order to look for additional incoming events matching the rest of components in the rules matched.

The main advantage of rule-based reasoning is that rules are expressed in a language easily understood by humans (statements "if...then"). However, characterizing attacks and expressing them manually in the form of rules requires a lot of effort and their quality depends on experts' ability. Moreover, the resultant rules are static and only suited to known attacks.

Statistical analysis can provide a solution for the detection of unknown attacks. Using statistical techniques we cannot be sure about the causality implications of links between different events, but we can explore unknown possibilities and we do not need to build a predefined model [10]. Adding the evaluation from a human expert to the results obtained by statistical methods can incorporate causality to the model.

III. Related Works

A. Event Correlation for Finding Multi-Step Attacks

The basis of our work is the consideration of attacks as an ensemble of events, an strategy composed by different steps. Many authors have considered this perspective.

Ebrahimi et al. [11] present a mechanism to dynamically extract attack scenarios from the correlation between alerts generated by an Intrusion Detection System (IDS). Marchetti et al. develop in [12] a pseudo-Bayesian algorithm where the previous alert history is used as a reference.

In [13] attack scenarios are built from a knowledge base of low level alerts obtained from the signatures in an IDS. The cause-effect relationship is deduced from the attacks received by each victim node. Following the inverse approach, the distributed system RIAC [14] deduces new attacks based on already defined prerequisites and consequences of minor alerts.

The order of sequences representing attacks is consider as an important feature in [15]. They cluster the alerts to study later their causal relationship using a Bayesian model.

All the works previously mentioned are only focused in alerts generated by an IDS and consider the construction of attack scenarios through the linking of these alerts.

Event correlation can also be tackled focusing in the abnormal character of attacks, as it is done by Friedberg et al. [16]. Their security system detects anomalies after learning from a test set clean of attacks. They have developed a whole mathematical framework for defining hypotheses, 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.

In other works, like [17], the focus is in fast retrieval of multi-step patterns. A tuple, the AC-Index, is updated after the retrieval of each sequence of alerts. Those sequences are extracted as non-contiguous sub-sequences in sets of events.

Finally, effort has been made for expressing the features of multi-step attacks. EDL is a signature language for representing them, first proposed by Meier [18] and later improved by Jaeger [7]. 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, as the use of tokens going through the network of nodes as search agents for indicating which events the system should look for.

B. Ant Colony Optimization

The algorithm presented here is based on the natural behaviour of foraging ants. In the foraging process we can observe a phenomenon of emergence as a complex behaviour arise from the individual actions of simple agents. Each ant modifies the environment deposing pheromones when they find a food source. Pheromones act as agents of indirect communication between ants in a process called stigmergy [19]. The cumulated pheromones create trails ants can follow for finding the food sources. Higher concentration of pheromones leads to shorter paths. Pheromones evaporation avoids trail stagnation to a sub-optimal path. The results got by the colony exceed so much the capabilities of an individual ant that it is difficult not to think there is a conductor managing it [20].

Ant Colony Optimization (ACO) [21] is a metaheuristic oriented to solve discrete optimization problems through indirect cooperation within a colony of artificial ants. The first ACO algorithm, Ant System, was introduced in 1991 [22] using the travelling salesman problem (TSP) as an example application, in the context of Dorigo's Ph.D. thesis [23]. Since then it has been successfully applied to many NP-hard optimization problems like optimizing traffic control signals [24] or scheduling a galvanizing line [25].

The bibliography about ant algorithms is plenty of papers trying to give answer to the problem of network intrusion detection. Several of them directly translate the ant metaphor to software agents moving through the network [26] [27].

However, if we consider the analysis of events in a central location, as we do in this paper, the most widespread technique is the anomaly detection through clustering of network data [28] [29] [30], in some cases combined with supervised methods such as SVM [31]. There are also some approaches introducing fuzzy systems [32] or relying in the importance of distributed clustering [33]. Other examples can be found in [34]. The limitations of all these centralized approaches is that they consider an attack as characterized by only one event, which is not always the case in real networks.

C. Manhill

An interesting version of ACO is the *Hommilière* (Manhill) system developed by Valigiani [35] in his Ph.D. thesis. It was created to be applied in an e-learning platform for recommending the best learning path to each user, according to the results obtained by other students. 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, deposing pheromones according to the student's success in each step [36].

Independently 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. [37] 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.

IV. Morwilog

Taking inspiration from ACO and Manhill, a system called Morwilog has been developed for linking sequences of attacks in a set of heterogeneous events. The prefix 'morwi-' is the translation to latin characters of the prefix meaning 'ant' in Proto-Indo-European [38], a theoretical reconstruction of the common ancestor of the Indo-European languages [39]. From now on, we give the name *morwi* to the artificial ants created during the execution of the proposed algorithm.

This 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. Furthermore, if the volume of events is too high for implementing an heuristic engine in real time, such mechanism can be used out of line for inferring general correlation rules that can later be imported into a regular correlation engine.

The main inspiration for Morwilog is the Manhill algorithm and how it links artificial ants to real entities. Our first hypothese was to directly translate to our case the way the algorithm is applied in e-learning, assigning a *morwi* to each user. Following this approach, users' behaviour would be profiled by the *morwis* so the system would learn from the actions of suspicious users. However, in the e-learning platform we have a limited number of students, while here the number of users can be very high if the network is connected to the Internet, as most are. Moreover, the frequency of connection can be very different from user to user, and most of them connect only once to the system. Finally, it is too simplistic to think an attacker is not going to hide his actions among regular traffic, making user profiling more difficult.

The real entity generating the artificial agent does not have to be necessarily a living entity, but it can be anything generated by one, and therefore unpredictable, as for example the events. Choosing the events as the origin of the *morwis* has the advantage that we can easily combine Morwilog with any event-oriented classification algorithm or apply event preprocessing for discarding irrelevant events.

Each relevant event retrieved by the system in the form of log could generate an artificial *morwi*, in the same way as processes are created in a classical correlation system. This *morwi* follows a path in a decision tree built with a representation of events as nodes. Each path counts with a level of pheromones, deposited by the *morwis* according to their success in detecting the attacks. This is evaluated by a security expert, who is in charge of reviewing alerts as it is done in a SOC (Security Operations Center).

A. General Definitions

The most important abstract construction we are working with is the event e, defined as an entity in the set of all possible events $\mathbb{E}$. We can consider an event $e = \{id_b/v_b, id_c/v_c, \ldots, id_a/v_a, \ldots\}$ as a finite set of components id_n/v_n, each of them representing an identifier/value pair storing information about specific aspects of the event. The system receives this information in the shape of a log. The identifier id_n defines an unique meaning for each value $e\,(id_n) = v_n$ in the event e. It establishes a correspondence with other events, which could also contain components with the same identifier and, therefore, the same meaning. Moreover, it univocally defines the type of value v_n of its pair, which could be a real number, an element from a finite set or an IP address, among others. We can find identifiers as "source", "timestamp" or "action", for instance.

We define a finite set of events ordered in time as $E = (e_1, e_2, \ldots, e_{N_E})$, with N_E the number of events. The set of events used as the input of our system is called 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 S_E. These sequences can eventually represent a multi-step threat. The set of all sequences representing a threat is denoted by A_E, with $A_E \subseteq S_E$. The task of a *morwi* is to find a sequence $s \in A_{E_{in}}$ starting the analysis from an event e_i.

Apart from the events themselves, we can define an abstract representation e^*, which corresponds to a subset of events through an ensemble of constraints. We call $\mathbb{E}^*$ to the set of all the possible e^*. We denote the set of all the possible abstract representations of event e_i as E_i^*. This set can be finite or infinite depending on the characteristics of the components represented in e_i. The operator $\odot$ is used for denoting the matching function between events and abstract representations, so $e_i \odot e^* = 1$ if $e^* \in E_i^*$, while $e_i \odot e^* = 0$ if $e^* \notin E_i^*$.

We can see e^* as composed by rules $e^*(id_n)$ to apply to the different components of the event for a finite set of identifiers id_n. Although most of the rules are considered as equalities (e.g. there is a match if component "protocol" is equal to "HTTP"), 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"). $e_i \odot e^* = 1$ only if $e_i(id_n)$ agrees with the rule defined in $e^*(id_n)\ \forall\ id_n$ represented in e^*.

B. The Stigmergic Scenario: the Tree

The *event tree* is the scenario where the pheromones are deposited and therefore where the stigmergic process takes place. Each node represents the profile of an event in the system and each tree is univocally defined by its root node. The *morwi* starts his search in the root node of the tree corresponding to the event that leads to its generation.

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 N_κ^j the total number of nodes in tree δ_j in a given moment and κ_0 the root node. The set of all possible nodes is denoted $\mathbb{K}$.

A tree has its nodes distributed in different levels. The node where the *morwi* starts its search when it is generated, κ_0, is followed by other levels of nodes representing events appearing later in time. The search of following events in the sequence will take place during a maximum time T_{max}, as the system will not have an unlimited amount of resources. The arcs connecting the nodes contain a particular level of pheromones, giving information to the *morwis* about which path they should choose with higher probability.

More precisely the nodes $\kappa_n = (e_n^*, F_n, \kappa_n^{(\alpha)})$ contained in a tree are composed by three elements:

- An abstract representation of an event $e_n^* \in \mathbb{E}^*$.
- A set F_n of N_f children $\kappa_m \in \delta_j$, each with an associated level of pheromones $\tau_{n,m} \in \mathbb{R}$. This means there is a level of pheromones associated to every link between nodes in the tree. 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 will be 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.
- A pointer to his ancestor $\kappa_n^{(\alpha)} \in \delta_j$, so pheromone propagation from deepest nodes to κ_0 can be made.

The element e_n^* allows the characterisation of a certain type of events the *morwi* is going to search in the set of events thanks to the operator $\odot$ defined previously. Actually, we can extend the operator $\odot$ for working between nodes and events, so $e \odot \kappa_n = e \odot e_n^* \forall \kappa_n \in \mathbb{K}, e \in \mathbb{E}$.

Events should be normalised to a common language before applying the operator, so the represented components mean the same in terms of security independently of event's origin.

The way the tree is defined depends on how the system is implemented, taking into account possible limitations such as system performance or structural simpleness. We have arranged the 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. An alternative option is to consider the tree as a sub-tree of a large hypergraph containing all the possible trees, so every *morwi* walks through the same structure. However, we have preferred the first approach because having the trees separated could ease the parallelization of the system.

C. Pheromone Evolution

According to the results returned by a *morwi* after traversing the tree, pheromones (τ) can be incremented for reinforcing a path leading to a real attack or decremented for penalizing the election of a node resulting in a false positive. Attacks are identified by a security expert from its knowledge of consequences and network's normal functioning. The change of pheromones is made by a certain amount called $\Delta\tau^+$ for the increment and $\Delta\tau^-$ for the decrement, with $\Delta\tau^+ > 0$ and $\Delta\tau^- < 0$. If we express as $\tau[n]$ the level of pheromones after update n, where $n \in \mathbb{N}$:

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

Furthermore, a mechanism of pheromone evaporation is needed for avoiding stagnation. The evaporation is made by decreasing the level of pheromones already present in the link between nodes by an evaporation rate ρ, being $\rho \in \mathbb{R}$ 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 is not applied to every link in the tree when there is an update, as there are nodes that the *morwi* cannot choose because during the detection process there is not any event matching those nodes. This does not mean they could have a high possibility to be leading to an attack, so we are avoiding the evaporation of pheromones associated to them for not penalizing rare events. Our system apply the evaporation only to the nodes in the same branch than the chosen path.

More generally, the modification of pheromones is applied directly only to those 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.

We introduce here an important variation with regard to classic ACO literature, where both increment and decrement

are a fixed amount independent of the level of pheromones. The combination with the evaporation leads to a variation of pheromones whose absolute value is higher at the beginning of the execution and decays as the system evolves and levels of pheromones are farther away from the initial value. The convergence to an upper limit of pheromones has been proved in [21]. However, we still want a higher decay in pheromone evolution for strongly penalizing bad chosen paths. Therefore, we introduce a dependency with $\tau[n]$ 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, fixed as a parameter. When a new tree is created, the pheromones of all his node-to-children links are initialized to this value. We have centered the gaussian function to $\tau[0]$, as we want a high 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.

We can experimentally prove that for certain values of the parameters the accumulation of pheromones has a natural upper limit in a path that always leads to an attack, which prevents the system to unlimitedly favor it. Following the same logic for $\Delta\tau^-$, we arrive also to 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. We also define τ_{min}^{atk} as the minimum level of pheromones for considering a path and attack.

D. The Algorithm

Before applying Morwilog to a set of events E_{in}, data should be classified according to the maximum search time T_{max} defined for the platform. We can define $E_{clas} = (e_b, e_c, \ldots, e_a, \ldots)$ as a subset of E_{in} 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}. Including other criteria of classification for creating E_{clas} allows to increment T_{max} preserving the performance of the system. Methods of data mining can be combined with deterministic filtering in the creation of E_{clas}. A good classification leads to a better detection of links between events.

Once the classification mechanism is defined, events can be classified as they are recollected, and each subset E_{clas} can be fed to the system when certain conditions, as number of events received, are met. It can be easily adapted for working in real time if we add a module for keeping track of the different subsets where the events are classified and we optimize system's idle time by allowing several *morwis* to work at the same time.

The sequence of actions performed by each *morwi* is summarized in pseudocode form in Algorithm 1. We are going to explain all the process carried out by the *morwi*, which begins when a relevant event e_i arrives to the system.

Algorithm 1 Morwi algorithm

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

$isresult \leftarrow$ 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 $\in S_{E_{clas}} \mid s_0 = e_i$
    $\delta_j \leftarrow$ 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_{clas}, i, F_n)
    if $C \neq \varnothing$ **then**
      $(\kappa_m, e_i, \tau_l) \leftarrow c \in C$, random τ-driven choice
      if $e_i = s'_0$ **then**
        Add branch to κ_n from s' with $\tau \leftarrow \tau_{min}^{atk}$
        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 $isresult$

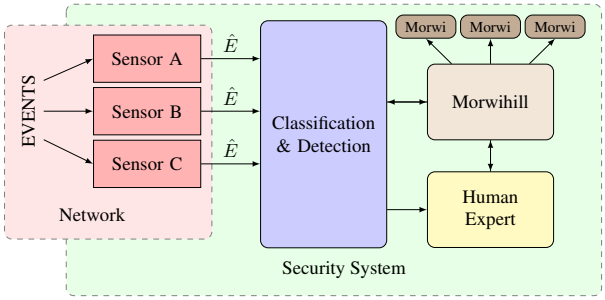

Fig. 1. Diagram of Morwihill integrated in a complete Security System under the supervision of a human expert. $\hat{E}$ = normalized events.

First of all, the *morwi* checks if there is already in Δ a tree δ_j with an upper node κ_0 such that $e_i \odot \kappa_0 = 1$. This is made by the function *find_tree*, which returns the tree we are looking for or an empty one ($\varnothing$) if it does not exist. If there is not any tree matching the event, with certain probability P_{tree} the

morwi forms a sequence from random events in E_{clas}. This sequence has e_i as its first element.

Algorithm 2 find_nodes algorithm

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

```
for each κ_m ∈ F_n do
    During time < T_max, find e_q | q > i, e_q ⊙ κ_m = 1
    Add (κ_m, e_q, τ_{n,m}) to C if e_q found
end for
if random(0,1) < P_jump then
    s' ← random ∈ S_{E_clas} | n > i ∀s_n ∈ s'
    Add ({e*_p, ∅, ∅}, s'_0, τ^atk_min) to C | s'_0 ⊙ e*_p = 1
else
    s' ← ∅
end if
```

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 $s = (s_0, s_1, \ldots, s_{L_s}) \in S_{E_{clas}}$ 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 of them traversed by the same path. The number of pheromones assigned to the links between nodes is $\tau[0]$. In equation 4 we can find the formal definition of the tree generated by *create_tree*, knowing that $s_p \odot e^*_p = 1, s_p \odot \kappa_p = 1 \ \forall s_p \in s$.

$$\delta_j = \{\kappa_p\} = \begin{cases} (e^*_0, (\kappa_1, \tau[0]), \varnothing) & p = 0 \\ (e^*_p, (\kappa_{p+1}, \tau[0]), \kappa_{p-1}) & 0 < p < L_s \\ (e^*_{L_s}, \varnothing, \kappa_{L_s-1}) & p = L_s \end{cases} \quad (4)$$

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

Once the *morwi* has C, 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^{atk}_{min} in each node-to-node link. In this case, this is the solution proposed by the *morwi*.

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 gets the returned result $s \in S_{E_{clas}}$ 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 last node in the sequence, according to Equation 2. Then, human feedback determines the nature of s introducing the component making reinforcement learning possible [40]. This is modelled by the function $\Pi(s)$, that returns 1 if s is an attack and 0 otherwise. Level of pheromones leading to last node is incremented if $\Pi(s) = 1$ and decremented if $\Pi(s) = 0$, following equations 1 and 3 so attack sequences are reinforced and innocuous one are penalized. Finally, changes in pheromones are propagated to the rest of the tree and the process is closed.

The integration of *Morwilog* in a complete security system can be observed in Figure 1. Once the classification of events and the detection of simple alerts are made, the *Morwilog* system, composed by the central *Morwihill* and the *morwis* generated by it, points out relevant relationships among events and alerts. New alerts are generated and represented in a console so the human expert can progressively evaluate them and mark them as attacks or not. This information is used to update the pheromones in the attack trees. Apart from that, *Morwilog* can send it back to the previous classification and detection subsystem, so it can improve its results.*Morwilog* is then the module introducing reinforcement learning into the whole security system.

Algorithm 3 Morwihill algorithm

Require: $E_{clas} \subseteq E_{in}$

```
for each e_i in E_clas do
    s, δ_j, isresult ← Morwi(E_clas, e_i)
    if isresult = true then
        (κ_0, ..., κ_l) path from δ_j matching s
        Evap. τ in paths, ∀κ_n ∈ δ_j with κ^(α)_n = κ^(α)_l
        if Π(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
```

V. Experiments

We have executed a set of experiments for evaluating the performance of our system. Our final aim is to apply *Morwilog* to a labelled dataset of heterogeneous logs where attacks are represented as composed by several events. It has to be 1) labelled for simulating the knowledge of the security expert, which we consider during the experiments as infallible. Moreover, 2) the events should come from a broad range of devices, as that is the type of data analyzed by SIEM nowadays and there are many attacks that have to be detected by their traces in different locations of the network. Finally, 3) the attacks have to be represented as a multi-step strategy, as single-step attacks can be detected by specific security systems

TABLE I
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

without the need of processing at the same time events from different sources.

Given the impossibility of finding an up-to-date dataset meeting simultaneously this three requirements, we have used an artificial dataset for this initial implementation of the system. The dataset has been created by Splunk Event Generator [41] is composed by a total of 1038 different types of logs. These types of logs are randomly taken by the generator in different proportions according to the type of event, and a random IP source is included. Timestamps are generated by a count, whose intervals of increment are also random. The standard audit trail format presented by Bishop [42] is used for representing the events in the database.

After logs are generated, sequences of events with certain IP source address are injected, representing 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 length of the sequences can be defined for each simulation.

The simulations have been carried out on an Intel Core i5 machine running at 1.4 GHz with 8GB RAM. We have repeated each simulation 50 times and we have taken the average results. The execution time grows linearly with the number of events to analyze. We have an average time of 1836 ms for a complete analysis of 40,000 events.

As the metrics for evaluating the system we have used the probability of detection (PD) and the probability of false alarm (PFA) as it is described by Marchette [43]. 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 the system.

We have nine parameters to determine before the execution of *Morwilog*. They are shown in Table I, with the values we have found more suitable to the dataset we were working with. The results of simulations are represented in Figure 2 in the shape of a ROC curve. PFA and PD are represented in horizontal and vertical axis, respectively. Each point correspond to the average result from 50 simulations with different sizes of dataset. 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. We have also

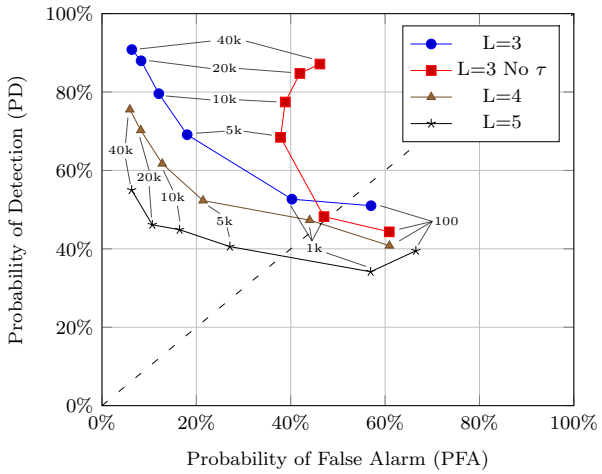

Fig. 2. ROC curve (PFA vs. PD) representing the results from simulations of *Morwilog*. Each point is the average of the result of 50 simulations with the same parameters (Table I) and random datasets (with different size of E_{in}, indicated next to the points). L is the length of injected sequences. "No τ" means simulations were made without pheromone update.

represented the results for a simulation with L=3 where the level of pheromones does not change but the parameters are kept the same ("No τ").

The results are clearly better as a higher number of events is analyzed, the curve getting closer to the upper left corner of the ROC plot. We also observe the positive effect of pheromones update, as when this is not done the PFA gets too high. Lastly, we observe that as the length of the sequences gets higher, the results are worse, as sequences are more intercalated with normal traffic.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented *Morwilog*, an ACO-based system for linking sequences of events that can be leading to an attack. The agents generated by the system walk through a set of trees following the pheromones left by other agents and accumulated in the links between nodes. The trees represent sequences of actions, where the paths representing an attack are finally outlined from the rest. Feedback is incorporated to the system after the evaluation of a human security expert, who validates if alerts represent actual attacks.

Preliminary simulations on an artificial set of logs generated by Splunk Event Generator [41] have been made for studying the parameters of the system and better developing the theoretical description of the algorithm. However, even if the first results seem positive, the lack of an heterogeneous set of events with labelled multi-step attacks did not allow to test it on a real environment. A sound definition of random attack trees and a good classification of events before applying *Morwilog* are fundamental for obtaining good results.

The next step is to test the system on a real dataset with the mentioned characteristics, after manual analysis for identifying and labelling multi-step attacks. This will lead to continue the development of the system shown in Figure 1 and will

allow to test different algorithms for anomaly detection and threat analysis. Within *Morwilog* itself, required improvements are the possibility of cloning ants when there are several suspicious paths or giving some meaning to the number of accumulated pheromones in a path so when it is too high the execution is stopped and an alert is returned. Finally, it is key to study the taxonomy of classic attacks for arriving to a more sophisticated way of building the trees.

ACKNOWLEDGMENT

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.

REFERENCES

[1] "SIEM. Security Information and Event Management software gives companies insight into both security and business operations," SC Magazine. Haymarket Media, Inc., apr 2016.

[2] "Standard on logging and monitoring," European Commission, 2010.

[3] J. Ya, T. Liu, H. Zhang, J. Shi, and L. Guo, "An automatic approach to extract the formats of network and security log messages," in *IEEE Military Communications Conference, MILCOM*. IEEE, 2015, Conference Proceedings, pp. 1542–1547.

[4] G. Suarez-Tangil, E. Palomar, J. M. De Fuentes, J. Blasco, and A. Ribagorda, *Automatic Rule Generation Based on Genetic Programming for Event Correlation*. Springer, 2009, vol. 63, pp. 127–134.

[5] M. Hasan, B. Sugla, and R. Viswanathan, "A conceptual framework for network management event correlation and filtering systems," in *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Management*. IEEE, 1999, Conference Proceedings, pp. 233–246.

[6] K. M. Kavanagh and O. Rochford, "Magic quadrant for security information and event management," Gartner, 2015.

[7] D. Jaeger, M. Ussath, F. Cheng, and C. Meinel, "Multi-step attack pattern detection on normalized event logs," in *IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, 2015, Conference Proceedings, pp. 390–398.

[8] A. Müller, "Event correlation engine," Master's Thesis, Eidgenössische Technische Hochschule Zürich, 2009.

[9] "Life cycle of a log," AlienVault, 2014.

[10] F. Alserhani, M. Akhlaq, I. U. Awan, A. J. Cullen, and P. Mirchandani, "MARS: Multi-stage attack recognition system," in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2010, Conference Proceedings, pp. 753–759.

[11] A. Ebrahimi, A. H. Z. Navin, M. K. Mirnia, H. Bahrbegi, and A. A. A. Ahrabi, "Automatic attack scenario discovering based on a new alert correlation method," in *IEEE International Systems Conference (SysCon)*. IEEE, 2011, Conference Proceedings, pp. 52–58.

[12] M. Marchetti, M. Colajanni, and F. Manganiello, "Identification of correlated network intrusion alerts," in *Third International Workshop on Cyberspace Safety and Security (CSS)*. IEEE, 2011, Conference Proceedings, pp. 15–20.

[13] Z. Zali, M. R. Hashemi, and H. Saidi, "Real-time attack scenario detection via intrusion detection alert correlation," in *9th International ISC Conference on Information Security and Cryptology (ISCISC)*. IEEE, 2012, Conference Proceedings, pp. 95–102.

[14] L. Zhaowen, L. Shan, and M. Yan, "Real-time intrusion alert correlation system based on prerequisites and consequence," in *6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*. IEEE, 2010, Conference Proceedings, pp. 1–5.

[15] R. Anbarestani, B. Akbari, and F. Fathi, "An iterative alert correlation method for extracting network intrusion scenarios," in *20th Iranian Conference on Electrical Engineering (ICEE)*. IEEE, 2012, Conference Proceedings, pp. 684–689.

[16] I. Friedberg, F. Skopik, G. Settanni, and R. Fiedler, "Combating advanced persistent threats: From network event correlation to incident detection," *Computers & Security*, vol. 48, pp. 35–57, 2015.

[17] A. Pugliese, A. Rullo, and A. Piccolo, *The AC-Index: Fast Online Detection of Correlated Alerts*. Springer, 2015, pp. 107–122.

[18] M. Meier, *Intrusion Detection effektiv!: Modellierung und Analyse von Angriffsmustern*. Springer-Verlag, 2007.

[19] G. Theraulaz and E. Bonabeau, "A brief history of stigmergy," *Artificial Life*, vol. 5, no. 2, pp. 97–116, 1999.

[20] D. M. Gordon, *Ant Encounters: Interaction Networks and Colony Behavior*. Princeton University Press, 2010.

[21] M. Dorigo and T. Sttzle, *Ant Colony Optimization*. MIT Press, 2004.

[22] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive feedback as a search strategy," Politecnico di Milano, Report 91-016, 1991.

[23] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. Thesis, Politecnico di Milano, Italy, 1992.

[24] S. Haldenbilen, C. Ozan, and O. Baskan, *An Ant Colony Optimization Algorithm for Area Traffic Control*. INTECH Open Access Publisher, 2013.

[25] S. Fernandez, S. Alvarez, D. Díaz, M. Iglesias, and B. Ena, *Scheduling a Galvanizing Line by Ant Colony Optimization*. Springer, 2014, pp. 146–157.

[26] G. A. Fink, J. N. Haack, A. D. McKinnon, and E. W. Fulp, "Defense on the move: ant-based cyber defense," *IEEE Security & Privacy*, vol. 12, no. 2, pp. 36–43, 2014.

[27] X. Hui, W. Min, and Z. Zhi-ming, "Using ant colony optimization to modeling the network vulnerability detection and restoration system," in *International Conference on Industrial Mechatronics and Automation (ICIMA)*. IEEE, 2009, Conference Proceedings, pp. 21–23.

[28] M. Kemiche and R. Beghdad, "CAC-UA: A communicating ant for clustering to detect unknown attacks," in *Science and Information Conference (SAI)*. IEEE, 2014, Conference Proceedings, pp. 515–522.

[29] D. P. Jeyepalan and E. Kirubakaran, "Agent based parallelized intrusion detection system using ant colony optimization," *International Journal of Computer Applications*, vol. 105, no. 10, 2014.

[30] G. Fernandes, L. F. Carvalho, J. J. P. C. Rodrigues, and M. L. Proena, "Network anomaly detection using IP flows with principal component analysis and ant colony optimization," *Journal of Network and Computer Applications*, vol. 64, pp. 1–11, 2016.

[31] W. Feng, Q. Zhang, G. Hu, and J. X. Huang, "Mining network data for intrusion detection through combining SVMs with ant colony networks," *Future Generation Computer Systems*, vol. 37, pp. 127–140, 2014.

[32] M. S. Abadeh and J. Habibi, "A hybridization of evolutionary fuzzy systems and ant colony optimization for intrusion detection," *The ISC International Journal of Information Security*, vol. 2, no. 1, 2015.

[33] M. N. K. Abdurrazaq, B. R. Trilaksono, and B. Rahardjo, "DIDS using cooperative agents based on ant colony clustering," *Journal of ICT Research and Applications*, vol. 8, no. 3, pp. 213–233, 2015.

[34] C. Kolias, G. Kambourakis, and M. Maragoudakis, "Swarm intelligence in intrusion detection: A survey," *Computers & Security*, vol. 30, no. 8, pp. 625–642, 2011.

[35] G. Valigiani, "Développement d'un paradigme d'optimisation par Hommilière et application á l'enseignement assistè par ordinateur sur internet," Ph.D. Thesis, Université du Littoral Côte d'Opale, 2006.

[36] G. Valigiani, E. Lutton, C. Fonlupt, and P. Collet, "Optimisation par "hommilière" de chemins pédagogiques pour un logiciel d'e-learning," *Technique et Science Informatiques*, vol. 26, no. 10, pp. 1245–1267, 2007.

[37] P. Mahanti, M. Al-Fayoumi, and S. Banerjee, "Simulating targeted attacks using research honeypots based on ant colony metaphor," *European Journal of Scientific Research*, vol. 17, no. 4, pp. 509–522, 2005.

[38] J. Pokorny, *Proto-Indo-European Etymological Dictionary*. Indo-European Language Revival Association, 2007. [Online]. Available: https://marciorenato.files.wordpress.com/2012/01/pokorny-julius-proto-indo-european-etymological-dictionary.pdf

[39] J. Clackson, *Indo-European Linguistics: An Introduction*. Cambridge University Press, 2007.

[40] D. K. Bhattacharyya and J. K. Kalita, *Network Anomaly Detection: A Machine Learning Perspective*. CRC Press, 2013.

[41] D. Hazekamp and C. Sharp. (2016, October) Splunk eventgen. Splunk. [Online]. Available: https://github.com/splunk/eventgen

[42] M. Bishop, "A standard audit trail format," in *Proceedings of the 18th National Information Systems Security Conference*. DTIC Document, 1995, Conference Proceedings, pp. 136–145.

[43] D. J. Marchette, *Computer intrusion detection and network monitoring: a statistical viewpoint*. Springer Science & Business Media, 2001.