Detecting Version Number Attacks in RPL-based Networks using a Distributed Monitoring Architecture

Anthéa Mayzaud
Inria Grand Est
Université de Lorraine
Nancy, France
Email: anthea.mayzaud@inria.fr

Rémi Badonnel
TELECOM Nancy
Université de Lorraine
LORIA UMR 7503
Nancy, France
Email: remi.badonnel@loria.fr

Isabelle Chrisment
TELECOM Nancy
Université de Lorraine
LORIA UMR 7503
Nancy, France
Email: isabelle.chrisment@loria.fr

***Abstract*—The concept of Internet of Things involves the deployment of Low power and Lossy Networks (LLN) allowing communications among pervasive devices such as embedded sensors. The IETF designed the Routing Protocol for Low power and Lossy Networks (RPL) for supporting these constrained networks. Keeping in mind the different requirements of such networks, the protocol supports multiple routing topologies, called DODAGs, built using different objective functions, so as to optimize routing based on several metrics. A DODAG versioning system is incorporated into RPL in order to ensure an optimized topology. However, an attacker can exploit this mechanism to damage the network and reduce its lifetime. In this paper we propose a detection strategy based on a distributed monitoring architecture with dedicated algorithms that is able to identify malicious nodes performing such attacks in RPL-based environments. The performance of this solution is evaluated through extensive experiments and its scalability is quantified considering a monitoring node placement method.**

I. Introduction

The growing interest for the Internet of Things has resulted in the large-scale deployment of Low power and Lossy Networks. They enable new applications ranging from smart electricity grids [1] to home automation solutions [2] [3]. The constrained devices composing these networks can be integrated with the existing Internet infrastructure, so that they exploit software services already available coupled in conjunction with their control and data gathering capabilities.

The Routing Protocol for Low-power Lossy Networks (RPL) was designed by the IETF RoLL working group to cope with resource constraints of embedded devices [4]. This protocol not only organizes nodes into DODAGs (Destination Oriented Directed Acyclic Graphs) but also optimizes the topology for application specific objectives, e.g. energy conservation, by using metrics and/or constraints available to a device. An RPL instance is a set of DODAGs, each with a specific objective function. Several RPL instances can be run within a network. A node can only join a single DODAG in one instance, however it can be part of several DODAGs only if they are in different instances. A node's rank value represents its position with respect to the DODAG root. This value always increases in the downward direction. To avoid rebuilding the entire DODAG when a parent node disappears, two *local repair* mechanisms are introduced by the protocol. The first one allows nodes to temporarily route through neighbors of the same rank, while the other one consists in using an alternative parent. It also provides a *global repair* feature to rebuild completely the DODAG. The mechanisms that enable RPL to provide this level of flexibility could also be manipulated by malicious nodes to harm the network.

In particular, the version number attack can misuse a RPL feature which is normally used for ensuring a loop and error free topology. A malicious node modifies the version number associated to a topology, thereby forcing a rebuild of the entire routing tree. Since the version number is included in control messages by parents, there is no mechanism provided by the standardized protocol to guarantee the integrity of the advertised version number. A forced rebuild can cause increased overhead, depletion of energy reserves, channel availability issues and even loops in the routing topology. Previous studies show that such attacks have a significant impact on RPL networks and highlight the importance of addressing them [5], [6].

We propose in this paper a solution based on a distributed monitoring architecture to detect version number attacks in RPL-based environments and identify the malicious nodes involved. Our main contributions are (1) the design of a detection strategy for these attacks and its associated algorithms, (2) the deployment and instantiation of the strategy using a distributed passive monitoring architecture, (3) the performance evaluation of our solution through extensive experiments and (4) the quantification of the proposed solution scalability in line with a node placement method.

The rest of the paper is organized as follows. An overview of related work is given in Section II. Section III presents the RPL protocol and the version number attack. The detection strategy relying on a monitoring architecture and detection algorithms is detailed in Section IV. The solution performance is analyzed

978-3-901882-85-2 © 2016 IFIP

in Section V, and a scalability evaluation is discussed in Section VI. Finally, Section VII concludes the paper and points out future perspectives.

II. Related Work

Version number attacks have already been analyzed in our previous studies such as [5] and [6]. We have shown that this type of attacks have severe consequences for RPL network causing significant control message overhead and building loops in the network which leads to a reduced lifetime for the network and the loss of data packets. Since many loops are built, the end-to-end delivery ratio decreases also. Authors of [6] have proposed to investigate these attacks under mobility with a probabilistic attacker model. They have also analyzed the impact of these attacks on nodes energy consumption and show that the batteries of the nodes were severely reduced by the attack. These analyses have confirmed the need for detecting and remedying these attacks.

In that context, the Version Number and Rank Authentication (VeRA) approach aims at preventing compromised nodes from impersonating the root and from sending an illegitimate increased version number [7]. It provides integrity of version numbers and ranks advertised in control messages via hash and signature operations. Their approach is shown to be faulty by the authors of [8] and [9], and another mechanism called TRAIL that uses the root as a trust anchor and monotonically increases node ranks is also proposed by them. Both approaches require maintaining state information on nodes that is likely to reduce already constrained computing resources.

Several intrusion detection systems have also been proposed for RPL-based networks [10], [11]. The first solution [10] is a specification-based IDS relying on a finite state machine implemented in each monitor node. The second IDS called SVELTE [11] is composed of three modules. One is responsible for rebuilding the topology at the sink nodes using requests, the second one carries out the intrusion detection process and the last one is a mini distributed firewall. However, none of them proposes to detect version number attacks. In order to prevent depleting the scarce resources of devices and provide a security-oriented monitoring solution, we propose a passive distributed monitoring architecture [12] which relies on higher order devices typically used in AMI[1] networks. We also prove that this solution is efficient to detect DAG inconsistency attacks. However, none of these solutions address version number attack properties.

III. Version number attacks in RPL networks

The RPL protocol is vulnerable to version number attacks, which exploit the *global repair* mechanism to overload the network. The root initiates a *global repair*, when too many inconsistencies are detected in the network. It consists in rebuilding the entire DODAG by incrementing the version number of the DODAG [4]. This number is carried in a type of control messages called DIO[2]. Each receiving node compares its existing version number against the one received from its parent. When the received version is higher, it must ignore its current rank information, reset trickle timers and initiate a new procedure to join the DODAG. This global repair mechanism guarantees a loop free topology, but is also quite costly. An older value of the version advertised in DIO messages indicates that the node did not migrate to the new version of the DODAG. Such a node should not be chosen as preferred parent by other nodes. Two versions of a DODAG can exist at the same time during a global repair. However, in order to avoid loops, data packets from the old version are allowed to transit in the new version but not the other way. As the convergence of the network has not been reached, the old version is no longer a DAG and loop free topologies cannot be guaranteed in this situation.

To avoid possible inconsistencies in the network, the version number should be propagated unchanged through the DODAG. However, there is no mechanism in RPL to ensure the integrity of the version number in received DIO messages. A malicious node may change this value in its own DIO messages to harm the network. Upon receiving a malicious DIO with a new version number, nodes reset their trickle timer, update the version and advertise this new version through DIO messages to their neighborhood as well. This can cause the illegitimate version number to propagate through the network. Such manipulation of the version number in the DIO packets causes both an unnecessary rebuild of the whole DODAG, and generates loops in the topology. Furthermore, since the new version of the DODAG is not built from the root, the topology is no longer acyclic, allowing loops to occur. This can negatively impact energy resources of the nodes, routing of data packets and channel availability. The pattern of this attack makes it difficult for a node to detect it locally. Indeed a malicious DIO packet coming from a parent seems to be legitimate for a node. When it comes from a child, then the node can consider this as due to an inconsistency in the network. Also localization of the malicious DIOs source is impossible in a purely local point of view, as nodes have only a view of their neighbors. They need to communicate with each other in order to find the origin of the attack.

IV. Detection Strategy

In the following section, we present our detection strategy for identifying version number attacks in RPL networks. This strategy uses our distributed monitoring architecture [12] on which we have deployed detection algorithms specifically designed for the version number attack. We first detail the architecture components and then describe the different algorithms that support this detection.

A. Security-oriented Monitoring Architecture

The strategy relies on a distributed monitoring architecture for the Internet of Things that passively observes the network based on dedicated nodes. This architecture uses RPL protocol mechanisms to perform monitoring and detection operations. Two types of nodes participating in the network compose

[1]Advanced Measurement Infrastructure

[2]DODAG Information Object

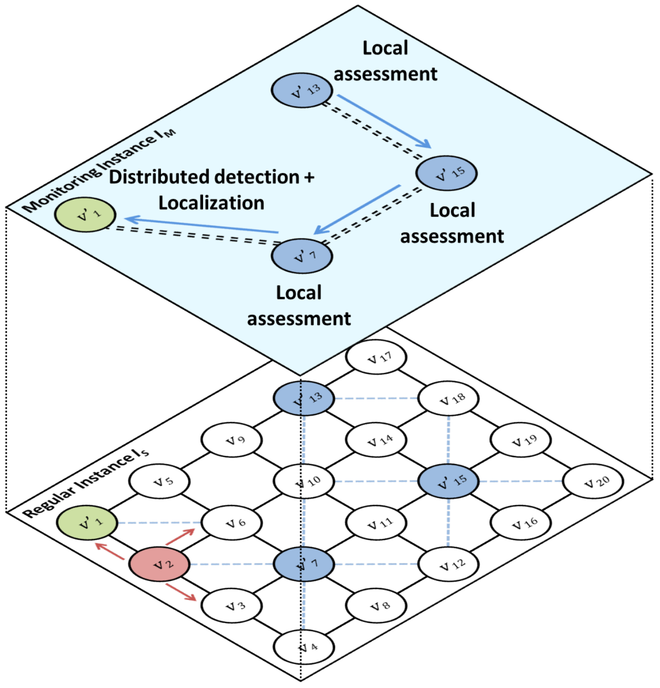


Fig. 1. Detection strategy based on our monitoring architecture.

this architecture, *monitored* nodes, also called regular nodes, plotted in white in Figure 1, and *monitoring* nodes plotted in blue which implements the detection solution.

The monitored nodes, noted $V = \{v_i\}$, are highly constrained devices that are typical C0 or C1 devices [13]. Their function is to carry out their sensing or actuation task and they constitute the so called regular network. The monitoring nodes ($V' = \{v'_k\}$) are on the other side, higher order devices that are at least C2 [13] or better. Since they have higher capabilities, they can perform monitoring and detection activities without impacting their ability to route information in the regular network. They are able to intercept and analyze packets sent by regular nodes and record necessary information. A monitoring node v'_k can only monitor its neighborhood $N_{v'_k}$ which is composed of all nodes in its communication range. However, network-level monitoring information is necessary to follow the topology and detect inconsistencies in the network such as maliciously incremented version number. As such, these monitoring nodes periodically forward the collected monitoring data towards the sink which can perform a distributed detection.

In order to preserve regular nodes resources, the monitoring nodes form a second routing topology, called *monitoring network*, as illustrated by the upper plane of Figure 1. This network has access strictly limited to monitoring nodes and is used to send collected information and results of detection algorithms. We exploit the multi-instance feature of RPL to build the two networks: one instance for the regular network noted I_R and one for the monitoring instance I_M. Figure 1 represents those two instances running at the same time. The two are completely independent which means that if the regular network breaks down at some point because of regular node failure or attacks, the monitoring network can still operate normally.

```
Algorithm LOCAL_ASSESSMENT
potential_att = NULL;
for each DIO received by $v'_k$ from $v_i \in N_{v'_k}$ do
    if ($VN_{v_i} > VN_{v'_k}$) and (potential_att == NULL) then
        potential_att = $v_i$
        send_root($M_k$ = ($VN_{v_i}$,$v_i$,$N_{v'_k}$))
    end if
end for
```

Fig. 2. LOCAL_ASSESSMENT algorithm implemented on monitoring nodes except the root.

B. Detection Algorithms

This section details the different algorithms used in our strategy for detecting version number attacks considering that only one attacker is present in the network at a given time. Due to the fact that an incremented version number is propagated in the entire graph, a monitoring node cannot decide by itself if this is the result of an attack or not. The monitoring nodes must share monitoring information to identify the malicious node. Our monitoring architecture is designed to allow monitoring nodes to collaborate together thanks to the monitoring instance network. Also, the monitoring nodes can track information regarding their neighborhood, so the regular nodes do not have to carry out this task. To detect an attack and identify malicious nodes, we propose detection and location algorithms described in Figures 2, 3 and 4. The LOCAL_ASSESSMENT algorithm presented in Figure 2 is deployed on monitoring nodes except the root and allows monitoring nodes to report to the root the sender of an incremented version number in their neighborhood. The algorithms presented by Figures 3 and 4 are implemented on the root node. The first one detects the attack and gather all monitoring node information into tables. The last algorithm performs the attacker identification by analyzing the collected information.

In the LOCAL_ASSESSMENT algorithm, a monitoring node v'_k, upon receiving a greater version number VN_{v_i} from v_i than its own version number $VN_{v'_k}$, sends to the root a message containing the address of the sender v_i and the list of its neighbors $N_{v'_k}$ obtained from the different received RPL control messages. The monitoring node only sends a message the first time it receives an incremented version number. Indeed, since the attacker is in the direct neighborhood of at least one monitoring node there is no need in sending further messages because senders of other incremented version number messages are relays. We also consider the other neighbors of the monitoring node as safe. Complementary to the algorithms, the root has the possibility to send a signal message indicating that the monitoring nodes should reset the *potential_att* value, in order to restart the detection process, in case another attacker appears in the network.

```
Algorithm DISTRIBUTED_DETECTION
anomaly_detected = 0
if ($VN_{v_j} > VN_{v'_1}$ in DIO received from $v_j \in N_{v'_1}$) then
   anomaly_detected = 1
   add(potential_att_list, $v_j$)
   add(neigh_list, $\{N_{v'_1}\}$)
   start(detection_timer)
end if
if ($VN_{v_i} > VN_{v'_1}$ in $M_k$ received)
and (anomaly_detected == 0) then
   anomaly_detected = 1
   start(detection_timer)
end if
while (potential_att_list.nb != $card(V')$)
or (!timer_expired(detection_timer)) do
   for each message $M_k$ received from $v'_k \in V'$ do
      add(potential_att_list, $v_i$)
      add(neigh_list, $\{N_{v'_k}\}$)
   end for
end while
LOCALIZATION()
```

Fig. 3. DISTRIBUTED_DETECTION algorithm implemented on the root.

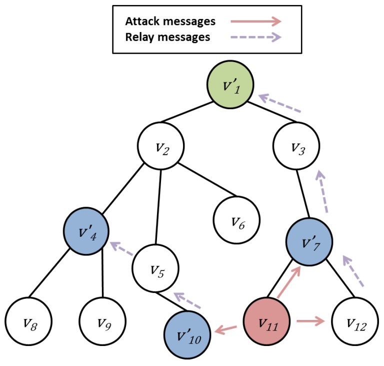
```
Algorithm LOCALIZATION
att_list = NULL
safe_list = NULL
for (i=0, i<potential_att_list.nb, i++) do
   if (att_list == NULL) then
      add(att_list,potential_att_list[i])
      add(safe_list,{neigh_list[i] $\setminus$ potential_att_list[i]}
   else
      if (potential_att_list[i] $\in$ att_list) then
         add(safe_list,{neigh_list[i]$\setminus$potential_att_list[i]}
      else if (potential_att_list[i] $\in$ safe_list) then
         add(safe_list,{neigh_list[i]$\setminus$potential_att_list[i]}
      else
         add(att_list,potential_att_list[i])
         add(safe_list,{neigh_list[i] $\setminus$ potential_att_list[i]})
      end if
      if (neigh_list[i] $\cap$ att_list $= v_m, v_m \neq \emptyset$) then
         remove(att_list,$v_m$)
      end if
   end if
end for
```

Fig. 4. LOCALIZATION algorithm implemented on the root.

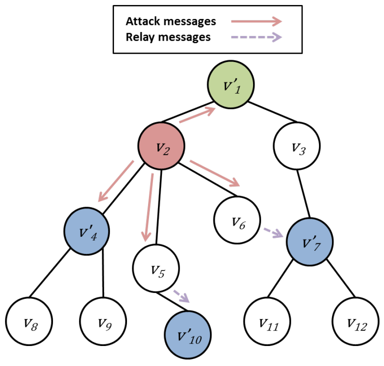
The DISTRIBUTED_DETECTION algorithm (see Figure 3 is supported by the root. Upon receiving either a monitoring message or an incremented version number, the root starts a detection timer to allow all monitoring nodes to send their messages. Two lists are managed by the root node: the $potential\_att\_list$ list which is composed of all v_i nodes reported by the different monitoring nodes and the $neigh\_list$ list which is composed of each monitoring node neighbors $N_{v'_k}$. Once the lists are completed, the root starts the localization procedure described in Figure 4.

This procedure exploits the two previous lists in order to produce two new lists: att_list list composed of nodes considered as malicious and the $safe\_list$ list containing all nodes classified as safe. The objective of this procedure is to compare neighborhoods of monitoring nodes in order to eliminate potential attackers. At initialization, the first element of the potential attacker list is added to the attacker list, and the other neighbors of the corresponding monitoring node are added to the safe list. While iterating, when the next potential attacker is already in the attacker list or in the safe list, it is ignored, and only the other neighbors are added to the safe list. This means that different monitoring nodes have detected the same node as a potential attacker, or that a monitoring node has detected a node as a potential attacker while being chosen as safe by another monitoring node. If the potential attacker is neither in *att_list* nor in the *safe_list*, it is added to the attacker list. The final test consists in verifying if some elements of the neighbor list are in the attacker list. This can happen when monitoring messages are received in a disordered manner. In this case, those elements v_m have to be removed from the attacker list. We can notice that at the end of the algorithm it is possible to obtain several nodes considered as attackers, when senders of incremented version number are monitored by only one monitoring node.

In order to illustrate these algorithms, we provide two examples describing the different possibilities using the topology presented in Figure 5. The first scenario shows our detection strategy functioning under normal conditions. The second scenario is used to present a use case where the detection strategy produces false positive results (normal node considered as malicious). In the first scenario (see Figure 5a), the attacker is located at position 11, it sends DIO malicious messages to all its neighborhood (plain red arrows) which are relayed by other nodes (in purple dotted arrows). The different monitoring nodes report to the sink the sender of abnormal messages and the list of their neighbors. At the end of this process, the potential attacker list and the neighbors list are established at the root as illustrated by Table I (a). Monitoring nodes v'_7 and v'_{10} receive attack messages from attacker v_{11} and send to the sink a message containing the sender of the anomalous message and their neighbors. Nodes v'_4 and v'_1 do the same with relays v_5 and v_3. Once all data is gathered, the sink can start the localization procedure to establish the list of attackers and the list of safe nodes. Table II (a) explains the different stages of those lists obtained by the localization procedure. At initialization, the first entry of potential attacker list, v_{11}, is added to the attacker list and the corresponding neighbors without the potential attacker $\{v_3, v_6, v_{12}\}$ are added to the safe list. Then since the second entry v_{11} is already in the attacker list, only the safe list is updated with the neighbors of monitoring node v'_{10} which are v_5 and v_9. The third entry is v_3 which is already in the safe list, so only the safe list is updated with the corresponding neighbors v_2. The same process is repeated for the last entry v_5 which is also already



(a) Attacker located at position 11.



(b) Attacker located at position 2.

Fig. 5. Version number attack illustrative examples.

TABLE I
POTENTIAL ATTACKER LIST AND NEIGHBORS LIST OBTAINED BY THE ROOT AFTER MESSAGES AGGREGATION.

(a) Attacker located at position 11.

Monitoring node	Potential attacker	Neighbors list
v'_7	v_{11}	v_3 v_6 v_{11} v_{12}
v'_{10}	v_{11}	v_5 v_9 v_{11}
v'_1	v_3	v_2 v_3
v'_4	v_5	v_2 v_5 v_8 v_9

(b) Attacker located at position 2.

Monitoring node	Potential attacker	Neighbors list
v'_1	v_2	v_2 v_3
v'_4	v_2	v_2 v_5 v_8 v_9
v'_7	v_6	v_3 v_6 v_{11} v_{12}
v'_{10}	v_5	v_5 v_9 v_{11}

TABLE II
STATES OF ATTACKER LIST AND SAFE LIST DURING THE LOCALIZATION PROCEDURE.

(a) Attacker located at position 11.

Step	Attacker list	Safe list
Initialization	v_{11}	v_3 v_6 v_{12}
Step 1	v_{11}	v_3 v_5 v_6 v_9 v_{12}
Step 2	v_{11}	v_2 v_3 v_5 v_6 v_9 v_{12}
Step 3	v_{11}	v_2 v_3 v_5 v_6 v_8 v_9 v_{12}

(b) Attacker located at position 2.

Step	Attacker list	Safe list
Initialization	v_2	v_3
Step 1	v_2	v_3 v_5 v_8 v_9
Step 2	v_2 v_6	v_3 v_5 v_8 v_9 v_{11} v_{12}
Step 3	v_2 v_6	v_3 v_5 v_8 v_9 v_{11} v_{12}

in the safe list. At the end of the algorithm, the only element of the attacker list is v_{11} which is correct and all the other regular nodes are considered as safe.

The second scenario, illustrated by Figure 5b, where the attacker is located at position 2 shows the case where the localization procedure produces two attackers. Table I (b) details how monitoring data is aggregated by the root v'_1 and Table II (b) shows the localization process. Until step 1 we can observe that only node v_2 is considered as the attacker, however in step 2, node v_6 is also added. The latter is not in the safe list meaning that no other monitoring node could exculpate him. As such, this detection algorithm may generate false positive results, a false positive corresponding to a normal node being detected as malicious by our strategy. In the next section, we will show that the number of false positives can be significantly decreased using a strategic monitoring node placement.

V. PERFORMANCE EVALUATION

We have evaluated the performance of our detection strategy through experiments by implementing a Proof of Concept prototype. The section details our experimental setup, the selection process for the placement of monitoring nodes and the the performance results of our detection algorithms.

A. Simulation Setup

In these experiments, we have set up a grid topology of 20 nodes corresponding to the lower plane of Figure 1. The grid topology was chosen because it allows relocation of the attacker to multiple positions easily, making it possible to study the performance of our detection strategy from different locations and neighborhood scenarios within a network. The Contiki 2.7 operating system was used to implement the sink, regular nodes and monitoring nodes. We have considered the attacker implementation proposed in [5]. The Cooja tool [14] was used to run the simulation with the compiled binaries of the different nodes. The radio model used was the DGRM model (*Directed Graph Radio Medium*) to emulate the links as shown in the lower plane of Figure 1: regular nodes can communicate with their neighbor horizontally and vertically while the monitoring nodes can also listen diagonally. Across all experiments, node v'_1 is the DODAG root, acting as the sink to which all other nodes send messages every twenty seconds to generate a background traffic. The attacker is designed to constantly send incorrect version numbers, which are greater than the root's version. Each simulation has lasted ten minutes which is enough to test our detection algorithms since only the first attack message is required for the detection as previously explained. The location of the attacker has been set to one of regular nodes, such that at least one simulation with the attacker located at every regular node is executed. This entire set of simulations is repeated three times for accuracy reasons. Attacks start after five minutes of simulation time, so that the network has enough time to settle and reaches a stable RPL topology. Not only the location of the attacker has been varied but also the location and the number of monitoring nodes. Indeed, we have seen in Section IV-B that it was possible to encounter false positives results depending on the fact that a node is monitored by one or several monitoring nodes. The next section details how and why different monitoring nodes configurations were chosen to evaluate the number of false positives.

B. Monitoring Node Placement Selection

Since the designed detection solution depends on the coverage of regular nodes by monitoring nodes, we defined the following metrics: (i) Cov_i representing the percentage of regular nodes covered by exactly i monitoring nodes ($i \in [1, M], M$ is the number of monitoring nodes); (ii) Ca_i representing the percentage of regular nodes covered by at least i monitoring nodes, e.g. $Ca_2 = Cov_2 + Cov_3 + Cov_4$ for $M = 4$.

In all cases, we target Ca_1 equals to 100% because all regular nodes should be covered by at least one monitoring node since the architecture is able to monitor all nodes. As shown by the second scenario in Section IV-B, Ca_2 is an important parameter for selecting the configurations to be considered, because the number of false positive depends on the neighborhood overlapping of the monitoring nodes. Therefore, monitoring nodes configurations have been selected for different Ca_2 values in order to quantify the impact of the Ca_2 value on the number of false positives. Five different Ca_2 values have been chosen including the lowest and the highest possible values for 4 and 5 monitoring nodes in the considered topology. The minimal number of required monitoring nodes is 4 so that Ca_1 equals to 100%. This value is given by the resolution of an Integer Linear Program (ILP) with our grid topology under the constraint that the sink, v'_1, is a monitoring node. The rest of the monitoring nodes are chosen among all the other nodes. A particular Ca_2 value corresponds to several combination of Cov_i. Therefore, one configuration of each combination has been chosen for the simulations. For 4 monitoring nodes, the number of possible configurations so $Ca_1 = 100\%$ is 24. We have also selected configurations with 5 monitoring nodes because the obtained Ca_2 values allow us to have zero false positive as illustrated by Figure 7b. For 5 monitoring nodes, 427 configurations can be run. We have simulated 29 configurations corresponding to the selected Ca_2 values. The different chosen Ca_2 values can be seen on Figure 7.

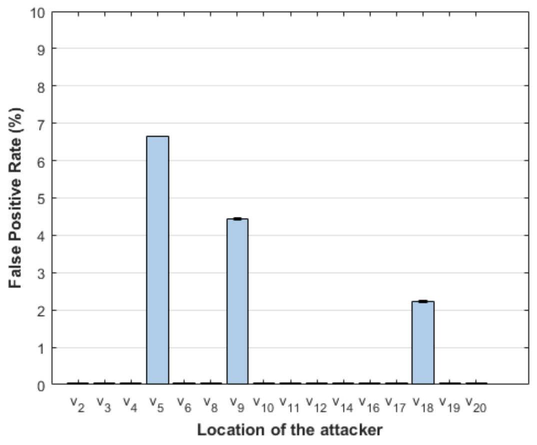


Fig. 6. False positive rates for different location of the attacker when configuration is the topology of 1.

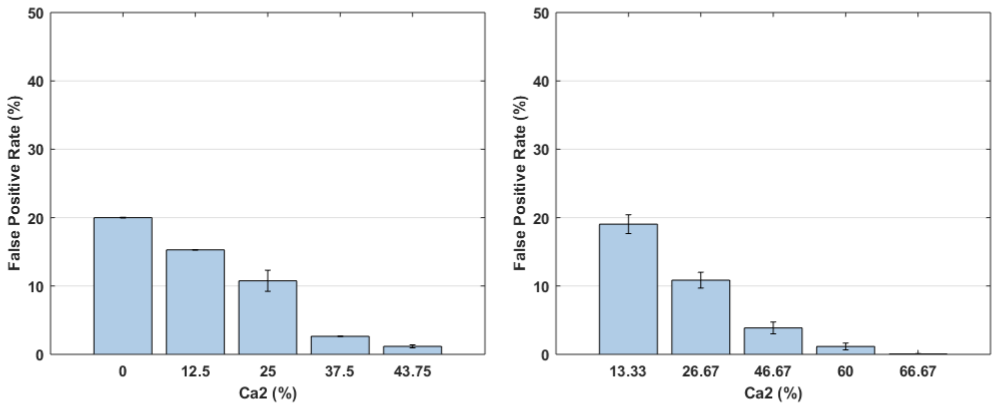
For each simulated scenario, the false positive rate (FPR) was calculated according to Equation 1, where FP and TN are respectively the number of false positives and the number of true negatives. A false positive is a node which has been incorrectly detected as malicious by our detection solution (the node is actually safe). A true negative is a node which has been properly considered as safe.

$$FPR = \frac{FP}{FP + TN} \tag{1}$$

C. Detection Results

Across all experiments, our detection strategy has successfully located the attacker, but other regular nodes were sometimes detected as malicious too. Figure 6 details false positive results for the topology presented in the lower plane of Figure 1 where the monitoring nodes are v'_1, v'_7, v'_{13} and v'_{15} and the Ca_2 is 43,75% (maximum value for 4 nodes). We can observe that the FPR is 0 for 13 positions of the attacker . Details about the detection results when the FPR is higher than 0 are given in Table III. When the attacker corresponds to node v_5, node v_9 is always detected as malicious too, because node v_9 is each time the direct relay of the attacker v_5 and is monitored by only one monitoring node (v'_{13}). No other monitoring nodes could have exonerate it. This is also the case for other positions of the attacker. However, attack relays were not considered as malicious each time. This can be explained by the fact that the attack relays can change depending on the timing for each simulation. For example, when the attacker is v_{18}, v_5 is considered as malicious only once, this is because monitoring node v'_1 receives only once the attack relay message from v_5. The other times, the relay node v_6 is also monitored by v'_7 which exonerates it. Similar results have been obtained for the other 35 configurations.

Figure 7 shows average false positive rate for the different values of Ca_2 when varying location of the attacker. Error bars are calculated as standard error of mean of the different possible configurations (Cov_i combination) for a particular Ca_2 value. Both figures show that the false positive rate decreases for increasing Ca_2 values, which means that the



(a) Configurations with 4 monitoring nodes. (b) Configurations with 5 monitoring nodes.

Fig. 7. Average false positive rate for different Ca_2 values.

TABLE III
DETAILED DETECTION RESULTS DETAILS RELATED TO FIGURE 6.

Attacker position	Series 1	Series 2	Series 3
v_5	v_5, v_9	v_5, v_9	v_5, v_9
v_9	v_5, v_9	v_5, v_9	v_9
v_{18}	v_5, v_{18}	v_{18}	v_{18}

more nodes are covered by at least two nodes, the less is the number of false positives. If we have 4 monitoring nodes we can observe in Figure 7a that the maximum value of the FPR is 20% which corresponds to the worst case (no node is covered by at least two monitoring nodes i.e. $Ca_2 = 0\%$), and at best the FPR stands around 1%. We obtain a false positive rate almost null when the Ca_2 is 66,67% (Figure 7b). Based on these results, we can conclude that monitoring nodes placement is clearly strategic in order to obtain satisfying performance in detecting version number attacks.

VI. Scalability Evaluation

We have also analyzed the scalability of our solution in line with the considered node placement. We have represented the problem of having at least $C\%$ of the nodes covered by at least two monitoring nodes using an optimization model. This constraint can be transformed into having at most (100 - C)% of the nodes covered by exactly one monitoring node which can be formulated as follows: for a given topology, a given connectivity matrix for all possible monitoring nodes placement in this topology, a given number of monitoring nodes and a given value C, find a configuration of monitoring nodes placement that minimizes the number regular nodes monitored by only one monitoring nodes so that at most (100 - C)% regular nodes are covered by strictly one monitoring node.

In that context, we have considered four parameters detailed in Table IV, as inputs to solve this problem. The first parameter is the size of the topology N. The second one is the connectivity matrix detailing the links of possible monitoring nodes with other nodes, $A_{i,j} = 1$ if node v_i can listen to node v_j. We set the diagonal of this matrix to 0, i.e. $\forall i, A_{i,i} = 0$ which means that we consider that monitoring node does not cover itself. The third parameter is the number of monitoring nodes M. The last parameter is the percentage of regular nodes we want to be monitored by at least two monitoring nodes C. The variables used are Y which represents if node v_i is monitoring node ($Y_i = 1$) or not ($Y_i = 0$), W indicating if node v_i is covered by monitoring node v_j ($W_{i,j} = 1$) or not ($W_{i,j} = 0$). The last variable is Z and represents if node v_i is covered by exactly one monitoring node ($Z_i = 1$) or not ($Z_i = 0$). The total number of variables for this problem is $N(N + 2)$.

TABLE IV
REQUIRED INPUTS FOR MONITORING NODES PLACEMENT.

Domain	Parameter	Description
$[\![1, N]\!]$	N	Number of nodes in the topology
$[\![1, N]\!] \times [\![1, N]\!]$	A	Connectivity matrix for monitoring nodes, $A_{i,j} = 1$ if node i covers node j
$[\![1, N]\!]$	M	Number of monitoring nodes
$[O, 1]$	C	Value indicating a percentage of nodes

TABLE V
CONSIDERED VARIABLES FOR MODELING.

Domain	Variable	Description
$[\![1, N]\!]$	Y	Binary variable indicating if Y_i is monitoring node (= 1) or not
$[\![1, N]\!] \times [\![1, N]\!]$	W	Binary variable indicating if node v_i is covered by monitoring node v'_j
$[\![1, N]\!]$	Z	Binary variable indicating if Z_i is monitored by exactly one monitoring node (= 1) or not

The constraints are detailed in Equations 2 to 7. Equation 2 is used to set v'_1, the root, as a monitoring node, it is possible to set another particular node to be a monitoring node according to the topology specifics. Equation 3 indicates how many monitoring nodes we choose. The constraint $Ca_1 = 100\%$ is given by Equation 4. Equation 5 calculates variable W which is used in Equation 6 to compute Z. The right part of this equation forces $Z_i = 0$ if v_i is a monitoring node or else $Z_i = 1$. The left part is equal to 1 only if $\sum_{j=1}^{N}(W_{i,j})$ is equal to 1 and v_i is not a monitoring node, which means that the left part equals 1 when the node v_i is monitored by only one monitoring node. Equation 7 indicates the constraint that at most (1-C) % of regular nodes are covered by exactly one monitoring node.

$$Y_1 = 1 \tag{2}$$

$$\sum_{j=1}^{N} Y_i = N \tag{3}$$

$$\forall i \in [\![1, N]\!] : \sum_{j=1}^{N}(A_{i,j}.Y_j) + Y_i \geq 1 \tag{4}$$

$$\forall i \in [\![1, N]\!] : W_{i,j} = A_{i,j}.Yj \tag{5}$$

$$\forall i \in [\![1, N]\!] : 2 - \sum_{j=1}^{N}(W_{i,j}) + 2.Y_i \leq Z_i \leq 1 - Y_i \tag{6}$$

$$\sum_{j=1}^{N} Z_i \leq (1 - C).(N - M) \tag{7}$$

The objective function f_{obj} given in Equation 8, is used to minimize the number of regular nodes covered by only one monitoring node i.e. to maximize the number of nodes covered by at least two monitoring nodes. This objective is necessary to compute the Z variable correctly. Indeed if v_i is not a monitoring node or a regular node only monitored by one monitoring node, Z_i can be equal to 0 or 1. Minimizing the sum on Z forces the default value to 0 in those cases.

$$f_{obj} = min \sum_{i=1}^{N} Z_i \tag{8}$$

We solved this problem with different sizes of grid topologies from 20 to 1000 nodes and with C=60% using the CPLEX solver [15] under the AMPL environment [16]. The C value was chosen according to previous results from Section V-C because the false positive rate was very low. A script was designed to establish the connectivity matrix of grids of corresponding sizes (4×5, 7×7, 10×10, 20×25, 25×40). The minimal number of monitoring nodes required to find a solution was determined empirically by running the solver several times. However the exploratory domain was restrained by solving a similar problem with the objective $Ca_1 = 100\%$ for every sizes. The model was also modified to find the minimal number of monitoring nodes so $Ca_2 = 100\%$. Figure 8 shows the minimal number of monitoring nodes required so Ca_2 is at least 60%. The value of C was set to 0.6 because it ensures low false positive rate for the detection algorithm,

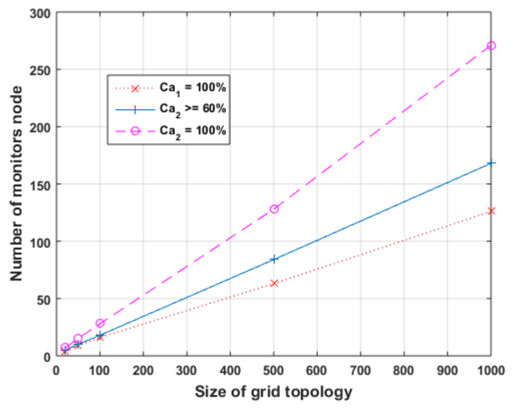


Fig. 8. Number of monitoring nodes needed to have $Ca_1 = 100\%$,$Ca_2 \geq 60\%$ and $Ca_2 = 100\%$ for different topology sizes.

as shown in Section V-C. We can observe that the number of monitoring nodes required to have the different Ca values is proportional to the number of nodes. However, the gradient is greater for larger values of Ca which means that we need more monitoring nodes for large size of grid. This results show that our solution supports scalability.

VII. Conclusions

We have proposed in this paper a detection strategy with dedicated algorithms to address version number attacks in RPL networks. We have instantiated this solution based on our distributed monitoring architecture which preserves constrained nodes resources. We have exploited monitoring nodes collaboration to identify the attacker, the attacker localization process being performed by the root after gathering detection information from all monitoring nodes. We have evaluated our solution through experiments and have analyzed the performance according to defined metrics. We have shown the false positive rate of our solution can be reduced by a strategic monitoring nodes placement. We have also considered the scalability issue by proposing an optimization problem which can be easily adapted to different topologies. By resolving this problem, we have quantified the number of required monitoring nodes to ensure an acceptable false positive rate for a given size of topology.

As future work, we are interested in performing complementary experiments in real infrastructures with additional classes of devices implementing the RPL protocol. We also want to evaluate and extend our solution to the case of attacker coalition where are several malicious nodes are involved at the same time in the network. We are also planning to enhance our architecture with other detection modules for addressing additional attacks [17].

Acknowledgment

This work was funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

References

[1] D. Popa, N. Cam-Winget, and J. Hui, "Applicability Statement for the Routing Protocol for Low Power and Lossy Networks (RPL) in AMI Networks," Internet Engineering Task Force, Internet-Draft draft-ietf-roll-applicability-ami-13, may 2016, work in Progress.

[2] E. Baccelli, R. Cragie, P. V. der Stok, and A. Brandt, "Applicability Statement: The Use of the Routing Protocol for Low-Power and Lossy Networks (RPL) Protocol Suite in Home Automation and Building Control," RFC 7733, feb 2016.

[3] M. Ersue, D. Romascanu, J. Schönwälder, and A. Sehgal, "Management of Networks with Constrained Devices: Use Cases," *IETF Internet Draft <draft-ietf-opsawg-coman-use-cases-02>*, July 2014.

[4] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, IETF, 2012.

[5] A. Mayzaud, A. Sehgal, R. Badonnel, I. Chrisment, and J. Schönwälder, "A Study of RPL DODAG Version Attacks," in *Proc. of AIMS conference*, 2014.

[6] A. Ahmet, S. F. Oktug, and S. B. O. Yalcin, "RPL Version Number Attacks: In-dept Study," in *Proc. of IEEE/IFIP Network Operations and Management Symposium*, Istanbul, Turkey, Apr. 2016.

[7] A. Dvir, T. Holczer, and L. Buttyan, "VeRA - Version Number and Rank Authentication in RPL," in *Proc. of the 8th IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, Hangzhou, China, October 2011, pp. 709–714.

[8] H. Perrey, M. Landsmann, O. Ugus, M. Wählisch, and T. C. Schmidt, "TRAIL: Topology Authentication in RPL," Tech. Rep., 2013.

[9] M. Landsmann, H. Perrey, O. Ugus, M. Wählisch, and T. C. Schmidt, "Topology Authentication in RPL," in *Proc. of INFOCOM. Poster*, 2013.

[10] A. Le, J. Loo, Y. Luo, and A. Lasebae, "Specification-based IDS for Securing RPL from Topology Attacks," in *Proc. of IFIP Wireless Days (WD)*, Niagara Falls, Canada, October 2011, pp. 1–3.

[11] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: Real-time intrusion detection in the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[12] A. Mayzaud, A. Sehgal, R. Badonnel, I. Chrisment, and J. Schönwälder, "Using the RPL Protocol for Supporting Passive Monitoring in the Internet of Things," Istanbul, Turkey, Apr. 2016.

[13] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," *IETF RFC 7228*, May 2014.

[14] "Cooja, the Contiki Network Simulator." [Online]. Available: http://www.contiki-os.org/start.html

[15] "IBM ILOG CPLEX Optimization Studio." [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html

[16] "A Mathematical Programming Language (AMPL)." [Online]. Available: www.ampl.com

[17] A. Mayzaud, R. Badonnel, and I. Chrisment, "A Taxonomy of Attacks in RPL-based Internet of Things," *International Journal of Network Security*, vol. 18, no. 3, pp. 459 – 473, May 2016.