A Simulation Tool for Evaluating Attack Impact in Cyber Physical Systems

Gianluca Dini, *Dept. of Ingegneria dell'Informazione/Res. Cent. "E. Piaggio", University of Pisa*, g.dini@iet.unipi.it, Italy
Marco Tiloca, *SICS Swedish ICT*, marco@sics.se, Sweden

Abstract

Security is getting an ever increasingly important issue in cyber-physical systems comprising autonomous systems. However, it is not possible to defend from all possible attacks for cost and performance reasons. An attack ranking is thus necessary. We propose a simulative framework that makes it possible to rank attacks according to their impact. We also describe a case study to assert its usefulness and effectiveness.

Keywords: Security, cyber-physical systems, risk assessment, simulation.

1 Introduction

Autonomous Systems results from the convergence of communication, computing and control. They are cyber-physical systems equipped with sensing, actuating and computing capabilities, interconnected through a wireless communication network, which may operate in both isolation and cooperation.

As with many of these complex networks of systems, it is possible for adversaries to intentionally compromise their functionality or performance. As to security, autonomous systems present two peculiarities with respect to conventional complex networks such as the Internet [1]. First a security infringement may traduce into a safety infringement with possible physical consequences. Second, autonomous systems are subject to both cyber and physical attacks. They are often deployed in open, unattended, possibly hostile environments where adversaries can physically attack them as well as interfere with the sensing process.

Security in autonomous systems is quite a new research field. Security vulnerabilities and related countermeasures are increasingly being discovered and exploited. However it is well-known that perfect security cannot be achieved for both performance and cost reasons. Thus, it is vital to define a threat model and then perform a risk assessment in order to determine the extent of potential threats and identify appropriate solutions. Typically, risk assessment involves two dimensions, namely the feasibility and impact of an attack. Here, we focus on the latter.

We present a simulation framework aimed at evaluating the impact of an attack in a cyber-physical system. The framework presents several advantages. First, it defines a simple *attack description language* that allows us to describe the effects of a cyber-physical attack in terms of *events* the attack generates. The language is composed of a reduced set of statements that make it possible to specify such events and thus easily describe even complex cyber-physical attacks such as a wormhole attack. Second, the simulator integrates an off-the-shelf simulator of the cyber-physical system under analysis and extends it as far as processing attack-related events is concerned. When the designer wishes to evaluate a new attack, he/she has only to provide the framework with the description of the attack. No line of code has to be re-written or part of the simular re-implemented. Third, and finally, the simulation framework makes it possible to *quantitatively* evaluate the impact of an attack, provided that appropriate security metrics have been defined. Methods and protocols for threat analysis have been defined, a recent relevant example being [5]. However, they tend tend to support a subjective analysis. This framework is a first stride towards a more objective analysis.

The paper is organizes as follows. Section 2 reports the main security requirements of autonomous systems. Section 3 provides an overall description of the simulation framework and its prototype based on Castalia [3], an off-the-shelf WSN simulator based Omnet++[7]. Section 4 discusses a case study where we apply our framework to analyze the impact of several attacks against the pollution monitoring system of an industrial plant. Finally, Section 5 reports our final conclusions.

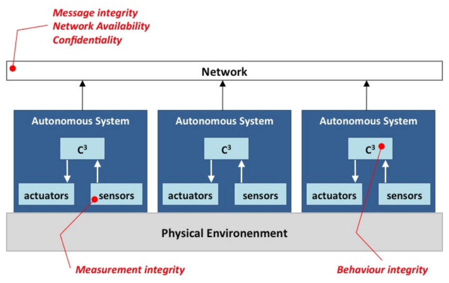

Figure 1: Autonomous Systems.

2 Security in Autonomous Systems

With reference to Figure 1, autonomous systems result from the convergence of communication, computing and control (C3). They are cyber-physical systems equipped with sensing, actuating and computing capabilities, interconnected through a wireless communication network, and operating in isolation and/or cooperation.

From a security standpoint, autonomous systems must fulfil the usual CIA requirements, namely confidentiality, integrity, and availability.

Informally, *availability* refers to the ability of authorized entities to act and collect data in a timely way. Availability guarantees that information is accessible and usable upon demand by the legitimate entity. In brief, availability guarantees that messages are received. A violation of network availability is a *denial of service*, i.e. the prevention of authorized access to data. Denial of service attacks are mainly based on network congestion and network jamming so that the network appears to be unavailable. In the former case, the system is actually busy in serving "fake" requests. In the latter case, an adversary maliciously creates interference with the radio frequency band used by the system by exploiting the broadcast nature of the wireless medium. Recently, selective jamming, a particularly insidious form of jamming aimed at a specific node, has been considered [4, 9].

Integrity refers to the confidence that actions are correct and collected data are accurate. A violation of integrity results in *deception*, a circumstance where an authorized entity receives false information about the phenomenon being monitored, and it believes it to be true (*data deception*). In addition, a deception may consist in one or more entities acting differently than specified (*actions deception*). In the most general case, an entity may be deceptive in terms of both data and actions. An integrity violation may have implications in terms of safety, loss of productivity or loss of reputation.

In computer security, the term integrity regularly refers to *message integrity*. However, this notion is limited and not sufficient to capture the integrity of the functional goal of autonomous systems. The interactions of autonomous systems with one another and with the physical world, together with the fact that the data sent by sensor nodes depends on their location, lead us to extend the notion of integrity by means of those of measurement and behaviour integrity.

Measurement integrity prevents the modification of a sensor measurement. An attack against measurement integrity succeeds when an autonomous system reports a sensed measurement that is not representative of the intended environment. A violation of the measurement integrity may derive from i) an *environment attack*, i.e. an attack affecting the environment around a sensor node by the adversary (e.g. placing a magnet on top of a magnetometer); ii) a *false position attack*, i.e. changing the location of a sensor node by the attacker, and the sensor node is unable to detect this change and report it; and iii) a *sensor spoofing attack*, i.e. sending the sensor a spoofed signal GPS spoofing [10] and ultrasound spoofing [1] are relevant instances of sensor spoofing attacks. This kind of attacks opens a new frontline that characterizes autonomous system security with respect to network security. Actually autonomous systems are designed and manufactured with certain safety measures. Once a system is manufactured and tested against natural errors, it is expected to conform to its design specifications unless accidental failures. However these safety measures are against non-malicious faults and thus usually do not consider mechanisms for adversary detection and prevention. As a consequence, the system safety may result fragile with respect to maliciously induced failures.

Behaviour integrity prevents the unauthorized modification of an autonomous system logic/behaviour. An attack against behaviour integrity succeeds when an autonomous system is compromised and does not behave as expected. The misbehaviour of a system manifests itself in reporting a fake measurement, sending a fake message, or taking a fake action. An adversary may violate the behaviour integrity of an entity by compromising it, reprogramming it in order to send incorrect data, perform wrong computations, or take wrong actions. This threat is particularly relevant in autonomous systems for several reasons. First, they are often deployed in unattended, possibly hostile environments. Second, often they are not equipped with physical protection mechanisms for cost and performance reasons. Third, autonomous systems are generally composed of several to many embedded computing units. Therefore, practical and functional reasons require a functionality of remote firmware updating. However, connecting embedded computing units over the network greatly increases the risk of attacks against behaviour integrity. Actually, if an attacker breaks into the remote firmware update channel, then he/she could compromise the security of the firmware by spoofing and counterfeiting it or even injecting a Trojan horse [6, 8].

Finally, *confidentiality* refers to the confidence that no information is disclosed to unauthorised principals. Confidentiality guarantees that information provided by an entity is accessible only to legitimate users. Privacy is a special case of confidentiality when the information is personal (e.g. information collected by a camera). A successful violation of confidentiality is called *disclosure*. A disclosure not only undermines sensitive and personal information but it can provide an adversary information for more effective attacks against integrity and availability. A disclosure attacks may be thus an intermediate step of a more elaborate attack strategy.

3 A framework for simulation of attack impact

The framework is composed of three components: i) an *Attack Description Language* that makes it possible to describe the *effects* of an attack; ii) an *Attack Simulator* that simulates the effects of attacks on the system under investigation and consequently makes it possible to evaluate their impact; iii) an *Attack Description Compiler* that convert attack effects descriptions into simulator configuration files.

The user first describes the effects of an attack to be evaluated by means of the Attack Description Language—possibly, descriptions are stored for later reuse. Then, the user compiles such a description into a configuration file which is provided as input to the attack simulator. Finally, the simulator simulates executions of the system affected by the described attack.

3.1 The Attack Description Language

The Attack Description Language (ADL) allows users to describe attacks to be evaluated. It is important to notice that here we are not interested in how an attack can be actually carried out. This issue attains to the feasibility of the attack which is the other dimension of risk assessment and is not our focus. Rather, we are interested in the effects of an attack once it has been successfully played. To fix ideas, let us consider an injection attack, a kind of deception attacks. We are not interested in how the adversary can inject fake messages in the system but, rather, in what are the effects of such messages once they have been successfully injected.

From such a standpoint, we assume that the successful execution of an attack produces a sequence of *events*, which takes place atomically. ADL consists in a collection of *statements* that allow the user to specify such a sequence of events. We consider two sets of *simple* statements: i) *node statements*, that allow us to describe alterations in node behaviour and account for physical attacks; and ii) *message statements*, that allow us to describe actions on network messages—including eavesdropping, injection, and dropping—and account for cyber attacks.

The node statements are

- destroy(nodeID, t) removes node nodeId from the network at time t.
- move(nodeID, pos, t) moves node nodeID to position pos at time t.
- spoof(nodeId, sensorId, value, t) returns the spoofed value value to sensor sensorID of node nodeID at time t.

The message statements are

- drop(pkt) discards the packet pkt.
- create(pkt, fld, content) creates a new packet pkt and fill its field fld with content.
- clone(srcPkt, dstPkt) clones packet srcPkt into packet dstPkt.
- change(pkt, fld, newContent) writes newContent into field fld of packet pkt.
- retrieve(pkt, fld, var) copies the content of the field fld of packet pkt into variable var.
- put(pkt, dstNodes, TX | RX, delay) puts packet pkt either in the TX or RX buffer of all nodes in the dstNodes list after a delay delay.

ADL provides other statements that allow *delayed* and *periodical* occurrence of events. For instance

```
schedule time = T; nodes = <list of nodes >
    {<list of events >}
```

and,

```
schedule delay = D; nodes = <list of nodes >
    {<list of events >}
```

specify that the list of events takes place on the list of nodes at time T or after delay D, respectively. In contrast

```
schedule time = T; period = P; nodes = <list of nodes >
    {<list of events >}
```

and

```
schedule delay = D; period = P; nodes = <list of nodes >
    {<list of events >}
```

specify that the list of events takes place periodically, with period P, on the list of nodes since time T or a delay D, respectively.

Finally, ADL allows us to describe the *conditional* occurrence of events. For instance

```
schedule time = T; nodes = <list of nodes >
    if(<condition >) {<list of events >}
```

specifies that the list of events takes place on the list of nodes if condition evaluates to TRUE.

The ADL makes it is possible to describe complex attacks in a concise although clear way. Let us consider a *wormhole attack* starting at time 200 s, and that tunnels MAC packets sent by node 3 to a remote area of the network containing nodes 15, 17, and 18. It follows that these nodes believe that node 3 is a neighbour of theirs whereas it is actually not. This attack may have severe implications on the integrity of the network because, if the tunnel stops delivering packets, the network gets partitioned. This attack can be described as follows

```
dstList=15,17,18;
schedule t = 200; nodes = "*"
    if(packet.MAC.source==3 &&
      packet.MAC.type==DATA)
        put(packet,dstList,RX,0);
```

It is worthwhile to notice that in the boolean condition we have used the dot notation packet.layer.field in order to specify the field field of packet packet in the header of layer layer. This means that, in general, the user must be aware of the actual specific network protocols that are in use at each communication layer, for each of them the packet header structures and fields, and finally the capabilities possibly offered by the simulator. For instance, the OMNeT++ platform [7] and the WSNs simulator Castalia [3] provide a set of objects, called *descriptors*, aimed at handling packets of a given communication layer and accessing their header fields.

3.2 The Attack Simulator Architecture

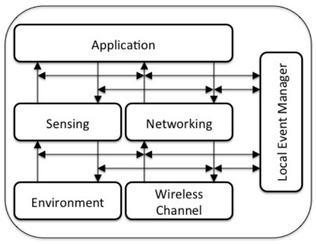

Figure 2: Attack Simulator: the Enhanced Node module.

With reference to Figure 2, the Attack Simulator considers every node as implemented by a *Enhanced Node* module which, in its turn, is composed of an Application Module, a Sensing Module, a Network module, and a Local Event Manager (LEM) module. All sub-modules but the LEM module can be off-the-shelf. The Application and Sensing modules may be composed of different sub-modules, which model the actual node application as well as physical sensing processes. Similarly, the Network module may include an arbitrarily complex combination of communication layers, e.g. routing and MAC.

The LEM module is devoted to the management of events related to attacks. LEM operates transparently with respect to the other components of the Node module. The LEM module intercepts all sensed data as well as network packets traveling through the communication stack. LEM may inspect and alter sensor data and packets content, add new ones, or even discard them. Also, it can alter the node behavior at different layers, change the node position in space, or even remove the node from the network.

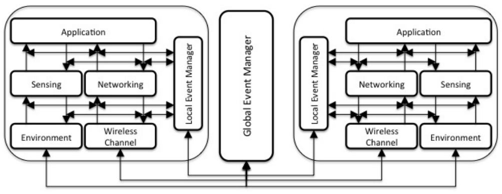

Figure 3: Attack Simulator: the Global Event Manager.

A system composed of several autonomous systems (also called nodes) is simulated by instantiating an Enhanced Node module for each node and a *Global Event Manager* (GEM) module that connects all the Enhanced Node modules. The GEM module is connected with every LEM module and allows them to communicate and synchronize in order to implement complex distributed attacks such as a wormhole attack, for example. Figure 3 shows the architecture of the simulator for a system composed of two interconnected nodes.

3.2.1 Reproducing events

A node may appear as argument of a node statement. In this case we say that the event specified by the statement occurs at the node. Similarly, a node may appear in the node clause of a delayed or periodical statement. In this case we say that the events listed in the delayed/periodical statement occur at the node.

At simulation startup, the simulator receives the attack configuration file, parses the attack statements and creates an event list for each node in the system. Let el_i be the event list of node i. The list contains all the events that occur at node i sorted in cronological order. We denote by el_i^j the j-th element in the event list el_i of node i. The simulator associates a timer to each element of the list. We denote by τ_i^j the timer associated to el_i^j. A timer is responsible to schedule the associated event.

Each element in the list specifies an amount of information that depends on the type of the associated event. If a list element is associated to a node event, the element stores the event type and the related actual arguments. If a list element is associated to a delayed event, then the element specifies the scheduling time, the associated node/message event and the related actual arguments. If an element is associated to a periodical event, the element specifies the scheduling time, the scheduling period, the associated node/message event and the related actual arguments. If conditional occurrence of events is present, the condition is stored in the element as well.

Whenever, a timer expires, the simulator consumes the associated event. In doing this, the simulator takes into account the possible associated condition. The simulator evaluates the condition to determine if the event has to be consumed or not. Furthermore, if the statement specifies a periodical event, the event is rescheduled according to the period.

3.3 A framework prototype for WSNs

We implemented a preliminary prototype of the framework for wireless sensor networks (WSNs). With reference to Figure 2, as to the Application, Sensing and Network modules we used Castalia [3], an off-the-shelf simulator for WSNs based on the discrete-event simulation platform OMNeT++ [7]. Castalia considers the network as a collection of nodes, which sense values according to a given physical process, and communicate through a commonly shared wireless channel.

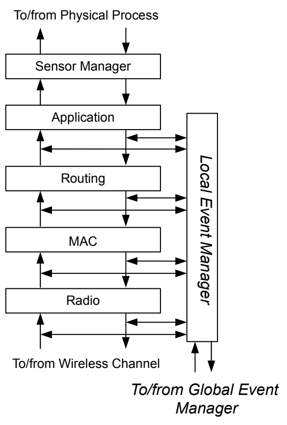

Figure 4: A Castalia-based prototype.

In the original architecture of Castalia, nodes are composed of different sub-modules. A sensor node application interacts with the physical process through a sensor manager module, and retrieves physical information from the environment. Furthermore, nodes are provided with a full communication stack, composed by a routing, a MAC, and a Radio layer. Thanks to such communication modules, the application sends/receives packets to/from the wireless channel. Also, Castalia provides the implementations of different routing and MAC layers.

In our implementation we integrated the Local Event Manager and the Global Event Manager within Castalia. With reference to Figure 4, in our preliminary implementation the Local Event Manager only intercepts incoming and outgoing packets traveling through the communication stack, between every pair of layers. In other words, the Local Event Manager does not intercepts sensor data and thus the simulation framework for the moment does not support the statement spoof.

4 A Case Study: Pollution Monitoring

4.1 Application scenario

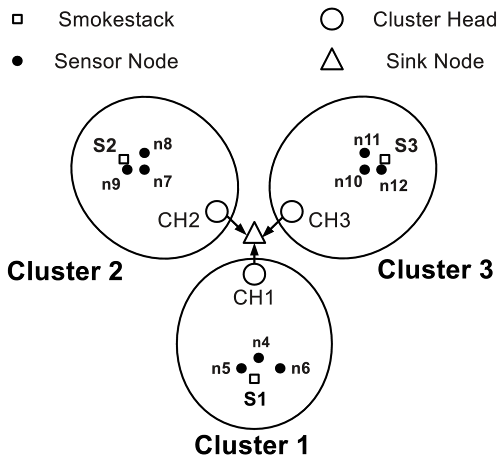

Figure 5: The industrial plant.

We consider a wireless sensor network (WSN) that monitors the pollution level of an industrial plant. With reference to Figure 5, we consider a plant comprising three independent smokestacks S1, S2, and S3, that release pollutant into the air.

A WSN has been deployed in the field, in order to monitor pollution levels. The WSN is organized in three clusters, C1, C2, and C3, one for each smokestack, respectively. Each cluster is composed of three sensor nodes and one *cluster head*. We denote by CH1, CH2, and CH3, respectively, the cluster heads of the three clusters. In each cluster, every sensor node periodically senses pollution emissions of the corresponding smokestack, and sends a report to its cluster head. This node periodically computes an average pollution level, and delivers it to a *sink* node.

The sink checks whether a single report exceeds a given threshold. The sink also aggregates reports from cluster heads to detect possible infringements of pollution limits at the level of the whole plant. So doing, possible anomalies, malfunctioning, or even conscious illegal deeds can be signaled to competent authorities.

Figure 6 shows the behaviour of the plant in an attack-free situation. Smokestack S1 infringes the pollution level limit, that we have supposed to be fixed at 37 $\mu g/m^3$. The dashed line depicts the pollution level

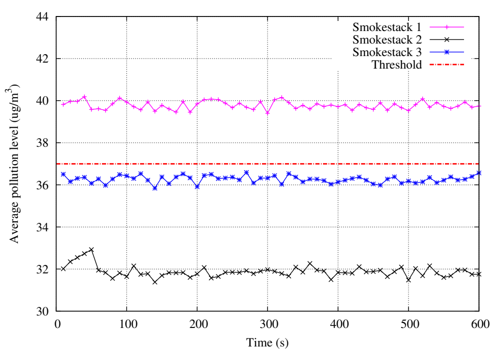

Figure 6: Attack-free system.

that smokestacks are supposed not to exceed. The other curves represent the average pollution levels over time for smokestack S1, S2, and S3. The graph shows that emissions from S1 exceeds the threshold.

Thanks to the WSN, it is possible to detect anomalies in pollutant emissions and react promptly. This of course assumes that sensor nodes and cluster head nodes work correctly, i.e. collected data are genuine and report delivery occurs regularly.

However, an adversary might have an interest to attack the WSN in order to tamper with the data collection process, alter the monitoring process, and mask the misbehavior of S1. In the next section, we discuss possible attacks against the WSN, specifically cluster C1, and evaluate their impact on the WSN overall monitoring capability.

4.2 The Threat Model

With reference to the application scenario described in Section 4.1, an adversary may compromise the monitoring service by altering reports produced by sensor nodes before being collected by cluster heads. In particular, an adversary might be interested in altering the computation of average pollution levels on cluster head CH1 in order to bring average pollution levels below the fixed threshold so that pollutant emissions from smokestack S1 would appear as regular, so concealing an actual limit infringement. In the following, we consider three possible attacks against the WSN, namely injection attack, misplace attack, and wormhole attack. The first attack is purely cyber, the second one is purely physical, whereas the third one is a cyber-physical one. Therefore, this attack selection provides the full range of attack types that can be launched against the WSN.

In an *injection attack*, the adversary creates fake report packets, and inject them into cluster C1, pretending they have been sent by a legitimate sensor node belonging to C1. Of course, fake values carried by such reports alter the computation of average pollution levels on cluster head CH1. This attack is quite hard to get detected. However, comparisons with other nodes' reports may help to contrast its effectiveness.

In a *misplacement attack*, the adversary captures one sensor node in cluster C1, and moves it from its original position to a new one. By properly choosing the new position, e.g. farther from smokestack S1 than the original, it is possible to make the sensor node measure a smaller value of the pollutant and thus alter the computation of average pollution levels on cluster head CH1. This attack is far more difficult to detect, since cluster head nodes assume that all sensor nodes' original positions remain unchanged over time.

In a *wormhole attack*, the adversary operates in two steps. First, the adversary captures one sensor node u from cluster C1, and places it in a different cluster in order to make the sensor node to measure pollutant emissions from a different but regularly emitting smokestack. Second, the adversary tampers the misplaced node u, in order to make it perform a wormhole attack. That is, node u does not send its report to the cluster head of the cluster where it has been moved to. Rather, node u forwards its report to cluster head CH1 through a dedicated low-latency channel. Therefore, as values reported by node u refer to a regular smokestack, the computation of average pollution levels of smokestack S1 by cluster head CH1 gets inevitably altered. It is well-known that wormhole attacks are particularly difficult to contrast.

4.3 Quantitative analysis of impact of attacks

In this section we report on the use of our simulation framework to quantitatively evaluate the impact of the attacks described in Section 4.2. We consider a WSN where each sensor node is equipped with a CC2420 radio chipset and runs the Multipath Rings routing protocol and the the T-MAC link-level protocol. We also assume that sensor nodes collect pollution measurements every 70 ms, while cluster heads compute average pollution levels every 10 s. Report packets transmitted by sensor nodes are 39 bytes in size, and include a payload whose size is 4 bytes. Finally, we assume the pollution level threshold is set to 37 $\mu g/m^3$. The adopted pollutant propagation model is based on the *Customizable Physical Process* provided by the Castalia simulator.

Simulation results has been obtained by means of 30 simulation runs, whose length was 600 seconds each. Each attack occurs at time $t = 200$ s.

4.3.1 Injection attack

In this attack, we consider an adversary injecting fake report packets into cluster C1. Specifically, the adversary creates fake report packets as follows. First, the value 4 is written in the source node ID field of each layer header. So doing, every forged packet appears originated from node $n4$ of cluster C1. Then, the report packet payload is set to 33 $\mu g/m^3$. Such a value is quite close to the average pollution level detected in cluster C2 (Figure 6) and thus results plausible from a cluster head CH1 standpoint. As a consequence, the attack is not easy detectable.

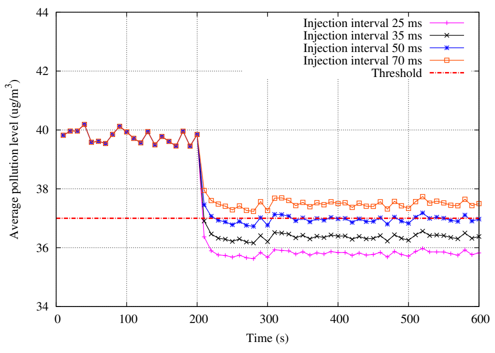

Figure 7: Pollution level reported under injection attack.

An important parameter of this attack is the adversary throughput. Of course the higher the throughput the higher the impact, but also the higher the visibility and therefore the risk of being detected. In order to evaluate the impact of the adversary throughput, we consider different *injection intervals* P_i. Figure 7 shows the effects of the injection attack for different values of P_i. As we can see, the larger the injection interval, the less effective the attack is. However, if $P_i < 50\ ms$, the attack is successfully performed, and the average pollution level goes beyond the threshold. Furthermore, it is evident that the adversary has no reason to perform the attack with an injection period smaller than 35 ms. In fact, it would require a great energy expenditure by the adversary, and could even be perceived as a Denial of Service, with increased chance of being detected.

4.3.2 Misplacement attack

In this attack, the adversary physically shifts $n5$ of cluster C1 away from smokestack S1. For the sake of simplicity, we assume that the node is only shifted along the y-axis only, towards cluster head CH1.

An important parameter of this attack is the displacement distance L. Figure 8 shows the impact of the attack for different values of L. As it turns out, shifting node $n5$ 2 m away from its original position is insufficient to mask over-emissions. Instead, if $L \geq 5$ meters, the adversary manages to achieve her objective. Of course, the farther sensor nodes are misplaced, the more effective the attack is.

Finally we observe that further simulative results showed us that the misplace attack is slightly less effective if it is sensor node $n4$ that gets misplaced. We omit these simulation results for the sake of brevity. However, we observe that without the simulation framework it might be difficult to estimate the attack impact by simply observing how sensor nodes are positioned in the field.

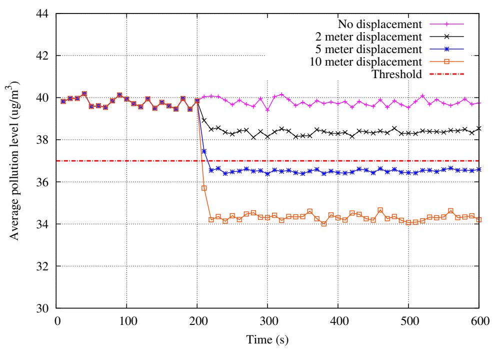

Figure 8: Pollution level reported under misplace attack.

4.3.3 Wormhole attack

In this attack the adversary shifts node $n5$ from cluster C1 into cluster C2, close to node $n7$. By doing so, node $n5$ measures pollutant emissions of smokestack S2 which, unlike smokestack S1, stays within an acceptable pollution level (see Figure 6). Then, sensor node $n5$ is reprogrammed in order to tunnel sensed data to cluster CH1 through a dedicated communication channel. Each sensed data is tunneled two times. Since the sample interval of $n5$ is 70 ms, and each sample is transmitted twice, we have an equivalent *wormhole period*, P_w, equal to 35 ms.

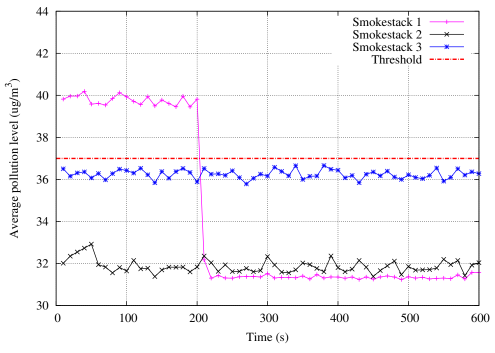

Figure 9: Pollution level reported under wormhole attack.

Figure 9 shows the effects of the wormhole attack on pollution monitoring. As we can see, the average pollution level in cluster C1 appears equal to about 31 $\mu g/m^3$, i.e. far below the threshold. This means the wormhole attack results to be even more effective than the injection attack discussed in Section 4.3.1. In fact, in case of injection attack with injection interval equal to 35 ms, cluster C1 displays an average pollution level comprised between 36 and 37 $\mu g/m^3$, that is closer to the threshold (see Figure 7).

4.4 Attack ranking

In this section, we describe a possible way to rank security attacks according to their impact.

4.4.1 A metric for integrity

In this section, we define a *security metric* aimed at measuring the respective impacts of the three attacks described and analyzed in Sections 4.2 and 4.3. Since the three deception attacks are against integrity, for the sake of brevity we limit ourselves to a metric that quantifies the level of deception of an attack. In the most general case, we should define metrics also for the levels of disclosure and denial of service. However, it is worthwhile to notice that wrong data is generally worse than no data, and therefore integrity is often given more priority than availability [2].

Notice also that computing security metrics requires to collect specific information, including the number and type of packets received by recipient nodes, the expiration of real-time deadlines, and the occurrence of packet interception by compromised nodes. In the most general case, application level information only might not be sufficient to compute security metrics. In other words, we also need information related to the network behavior and the actual communication among nodes. Therefore, unless the considered system model relies on very strong, and possibly unrealistic, assumptions, we believe that network and attack simulation is a valuable and essential approach to gather essential information, and perform the attack ranking process.

Let x_A denotes the level of deception of an attack A. That is, x_A indicates how information v' in the presence of attack A differs from information v when the system is attack free. Such metric is an extension of those described by Cardenas *et al.* in [2]. Formally,

$$x_A = \frac{\sum_{i=1}^{M} r_i \cdot \left(\frac{\sum_{j=1}^{S_i} \|v_j - v'_j\|^2}{S_i} \right)}{M} \tag{1}$$

where M denotes the number of devices handling forged samples; r_i the weight of the i-th device; S_i the number of samples considered on the i-th device, and, finally, v_j and v'_j the expected and the forged j-th sample, respectively. Notice that although the adversary has no particular restrictions on the choice of v'_j, however she has to be careful that v'_j is still valid from a system standpoint.

4.4.2 Attack severity evaluation

With reference to Equation 1, for each attack A, we compute two values: i) x_A^C, which refers to the impact of the attack on cluster C1, i.e. it takes into account the average values computed by cluster head CH1; and ii) x_A^S which refers to the impact on the system as a whole, i.e. it takes into account average values computed by the sink node. In both cases, we assume $M = 1, r_1 = 1,$ and $S_1 = 60.$. As to the injection attack, we consider several injection intervals, namely, 25, 35, 50, and 70 milliseconds. As to the misplacement attack, we consider several displacement distances, namely 2, 5, and 10 meters. Finally, as to the wormhole attack, we consider several wormhole intervals, namely, 23.3, 35, and 70 milliseconds.

Position	x_A^C	**Attack**	**Attack parameter**
#1	57.203	Wormhole	$P_w = 23ms$
#2	47.159	Wormhole	$P_w = 35ms$
#3	31.655	Wormhole	$P_w = 70ms$
#4	19.735	Misplace	$L = 10m$
#5	10.493	Injection	$P_i = 25ms$
#6	7.751	Injection	$P_i = 35ms$
#7	7.048	Misplace	$L = 5m$
#8	5.297	Injection	$P_i = 50ms$
#9	3.504	Injection	$P_i = 70ms$
#10	1.332	Misplace	$L = 2m$

Table 1: Rank of attacks at cluster level.

Position	x_A^S	**Attack**	**Attack parameter**
#1	7.056	Wormhole	$P_w = 23ms$
#2	5.906	Wormhole	$P_w = 35ms$
#3	4.168	Wormhole	$P_w = 70ms$
#4	1.135	Injection	$P_i = 25ms$
#5	0.921	Misplace	$L = 10m$
#6	0.833	Injection	$P_i = 35ms$
#7	0.564	Injection	$P_i = 50ms$
#8	0.499	Misplace	$L = 2m$
#9	0.389	Misplace	$L = 5$
#10	0.369	Injection	$P_i = 70ms$

Table 2: Rank of attacks at system level.

Table 1 and 2 report the attack rank according to the computed value of x_A^C and x_A^S, respectively. Since the adversary thwarts cluster C1 activities, the impact of each attack is more severe from a cluster standpoint rather than from a whole system point of view. Also, the wormhole attack always displays the most severe impact against integrity. This is because the misplaced node reports a value that is genuinely small although belonging to another cluster. Of course, the higher the wormhole transmission rate, the higher the attack impact. Finally, integrity in cluster C1 is more affected by the misplacement attack, while the injection attack results to be more effective from a whole system standpoint.

4.4.3 Discussion

Unfortunately space prevents us from discussing countermeasures to the above attacks. However, we would like to give at least a few intuition about how the simulation framework may help a designer to choose countermeasures that implement the best trade-off between cost and efficacy.

As to the injection attack, we can envision two possible countermeasures. The first one consists in authenticating packets. While it is highly effective against an injection attack, it has two disadvantages. First, it introduces the problem of key management. Furthermore, it requires to update the software onboard sensor nodes in order to support this security control. Second, it causes an enlargement of packets because of the attached authenticator. Such an enlargement would increase communication overhead. The other possible countermeasure consists in increasing the redundancy by deploying additional sensor nodes. Intuitively, fake packets would weight less in percentage. The advantage of this solution is that it does not require to install any software, does not require any key management, and does not cause any packet enlargement. On the other hand, by increasing the number of nodes, the overall network traffic would increase as well. The simulation framework could be used to evaluate the cost in terms of additional network traffic of each countermeasure.

Notice also that the application has a reporting period of $70ms$ whereas an injection attack has a meaningful impact at the cluster level (Table 1) if the attack injection period is comprised between 35 and $50ms$. This information could be useful to setup an intrusion detection system to suspect anomalous traffic rates that are $1.4 \div 2.$ times as expected.

As to the misplacement attack, possible solutions could be secure data aggregation or secure localization. Another possible solution could be some form of physical protection of nodes, in order to physically prevent an adversary from shifting a sensor node from its established position. However, the analysis carried out by means of the simulation framework allows us to realize that it is not necessary to physically protect all the sensor nodes. As it turns out from Figure 8, physical protection is only necessary for those sensor nodes that are close to smokestack S1 (say, less than $30m$).

As to the wormhole attack we state that packet authentication would be useless. Actually, the misplaced node $n5$ would use the correct keying material to authenticate packed carrying data sensed in another cluster. Notwithstanding, increasing node redundancy would remain an option. The simulation framework would allow us to evaluate both the efficacy of this options as well as its cost in terms of traffic increment. This option should be compared to alternative options based on secure localization and/or physical protection.

5 Conclusions

Security is an important design dimension in cyber-physical systems including autonomous systems, because it is conducive of safety infringements. However, in practice it is not possible to address all possible attacks. For this reason, we need a tool that allows a designer to determine the most "important" ones. We have presented a simulation framework especially conceived to support the designer in this task. The framework allows a designer to describe and simulate attacks, quantitatively evaluate their effects, and, finally, rank them according to such effects. The framework provides a simulation description language that makes it possible to define reusable attack descriptions in a relatively simple way. Furthermore, the framework support and promotes reusing of off-the-shelf simulators. For the moment the framework has been integrated in Castalia, a WSN simulator based on Omnet++.

Acknowledgements

This work has been partially supported by the EU FP7 Project PLANET, "Platform for the Deployment and Operation of Heterogeneous Networked Cooperating Objects" (`www.planet-ict.eu`), and by the Italian PRIN Project TENACE, "Protecting National Critical Infrastructures From Cyber Threats" (`www.dis.uniroma1.it/~tenace/`).

References

[1] K.D. Akdemir, D. Karakonunlu, T. Padir, and B. Sunar: *An Emerging Threat: Eve Meets A Robot* Proceedings of the Second International Conference on Trusted Systems (INTRUST 2010), LNCS 6802, pp. 271–289, 2011.

[2] A.A. Cardenas, T. Roosta, and S. Sastry: *Rethinking security properties, threat models, and the design space in sensor networks: A case study in SCADA systems*. Ad-Hoc Networks, vol.7, no.8, pp.1434-1447, November 2009.

[3] National ICT Australia: *Castalia*. Available at `http://castalia.npc.nicta.com.au/`.

[4] R. Daidone, G. Dini, M. Tiloca: *A solution to the GTS-based selective jamming attack on IEEE 802.15.4 networks*, Wireless Networks, 2013.

[5] European Telecommunications Standard Institute: *Telecommunications andf Internet converged Services and Protocols for Advanced Networking (TISPAN); Methods and protocols; Part 1: Method and proforma for Threat, Risk, Vulnerability Analysis*, ETSI TS 102 165-1 V4.2.3 (2011-03), 2011.

[6] C. Gorog: *Protect Firmware from Counterfeating*, `http://www.embeddedintel.com/special_features.php?article=1265`, 2011

[7] *OMNeT++ Network Simulation Framework* Available at: `http://www.omnetpp.org/`.

[8] L.K. Shade: *Implementing Secure Firmware Updates* Proceedings of the Embedded Systems Conference Silicon Valley, 2011.

[9] M. Tiloca, D. De Guglielmo, G. Dini and G. Anastasi: *SAD-SJ: a Self-Adaptive Decentralized solution against Selective Jamming attack in Wireless Sensor Networks*. In Proceedings of the IEEE International Conference on Emerging Technology & Factory Automation (ETFA 2013), Cagliari (Italy), 2013

[10] J.S. Warner and R.G. Johnston: *GPS spoofing countermeasures* Homeland Security Journal, 2003.

[11] A.M. Wyglinski, X. Huang, T. Padir, L. Lai, T.R. Eisenbarth, and K. Venkatasubramanian: *Security of Autonomous Systems Employing Embedded Computing and Sensors* IEEE Micro, vol.33, no.1, pp. 80–86, January/February, 2013.