Simulative evaluation of security attacks in networked critical infrastructures

Marco Tiloca[1], Francesco Racciatti[2], and Gianluca Dini[2]

[1] SICS Swedish ICT AB, Security Lab
Isafjordsgatan 22, 16440 Kista (Sweden)
marco@sics.se
[2] Dept. of Ingegneria dell'Informazione, Univ. of Pisa
Largo Lazzarino 1, 56100 Pisa (Italy)
f.racciatti@studenti.unipi.it, g.dini@iet.unipi.it

Abstract. ICT is becoming a fundamental and pervasive component of critical infrastructures (CIs). Despite the advantages that it brings about, ICT also exposes CIs to a number of security attacks that can severely compromise human safety, service availability and business interests. Although it is vital to ensure an adequate level of security, it is practically infeasible to counteract all possible attacks to the maximum extent. Thus, it is important to understand attacks' impact and rank attacks according to their severity. We propose SEA++, a tool for simulative evaluation of attack impact based on the INET framework and the OMNeT++ platform. Rather than actually executing attacks, SEA++ reproduces their effects and allows to quantitatively evaluate their impact. The user describes attacks through a high-level description language and simulates their effects without any modification to the simulation platform. We show SEA++ capabilities referring to different attacks carried out against a traffic light system.

Keywords: Security, attack simulation, OMNeT++, INET

1 Introduction

ICT is a fundamental component in monitoring and controlling critical infrastructures (CIs) such as electricity, railway and traffic systems. CIs are essential in the proper functioning of our daily life and their security is extremely important. In fact, a security infringement may have severe adverse consequences in terms of human being safety, service availability and business interests. In the past, CIs were somewhat secure as they had limited connectivity. However, the increased connectivity to the Internet and the corporate network, as well as

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. The research leading to these results has received funding from the *European Union* Seventh Framework Programme (*FP7/2007-2013*) under *grant agreement* n° 246016.

the use of commodity hardware, off-the-shelf protocols and software components make CIs no longer immune to cyber-security attacks.

In order to better understand the protection of CIs, it is important to analyze the security risks of such systems and develop appropriate solutions to protect them from malicious attacks. Unfortunately, addressing all the possible attacks is not viable, either from a practical or from an economical viewpoint. It is thus necessary to identify the attacks that have a more severe impact and focus on them. A possible approach to achieve this goal is via *simulation*. Simulations are important due to the fact that it is impractical to conduct security experiments on a real system, because of the scale and the cost of implementing standalone systems, as well as the potential risk of system downtime. On the other hand, although well consolidated, an analytical approach based on system theory does not provide a complete modeling of the ICT infrastructure [13].

In this paper, we present SEA++, a simulation tool aimed at quantitatively evaluating the impact of security attacks against the ICT infrastructure of a CI. We consider both cyber and physical attacks, where the former are addressed to messages, whereas the latter are addressed to nodes composing the infrastructure. A distinctive feature of our tool is that it allows us to simulate the effects of an attack by reproducing the events that the attack generates. This implies that we do not need to implement or port an attack, with clear advantages in terms of analysis time.

The tool is based on an off-the-shelf network simulator that we extend, but not modify, by integrating components for the processing of attack events. Good simulators are always the result of a large effort, and therefore any modification is preferably avoided. In particular, we use the INET Framework, an open-source model library for the OMNeT++ simulation environment, that contains networking models including those for the Internet stack, wired and wireless link layer protocols, and mobility [1][2]. Finally, our tool is also flexible, in that it allows us to describe attacks by means of a simple *attack specification language*. In order to simulate the effects of an attack, it is sufficient to provide a description of the events that it generates in that language.

The rest of the paper is organized as follows. In Section 2, we discuss related works. In Section 3, we illustrate the main concepts behind the simulation tools, briefly introduce the attack description language and sketch the simulator architecture. In order to show the tool capabilities and potentialities, in Section 4 we discuss a case study based on attacks against a traffic light control system. Finally in Section 5 we draw our conclusions.

2 Related work

A number of different approaches to attack impact analysis have been presented so far. For instance, [10][12][15] discuss analytical models aimed at detecting and contrasting attacks, and rely on simulation to validate their own correctness and efficiency. In [3], Genge *et al.* presented AMICI, an assessment/analysis platform for multiple interdependent critical infrastructues. AMICI relies on simulation

for the physical system components and an emulation testbed based on Emulab to recreate cyber components [5].

Wang and Bagrodia proposed SenSec, a framework that simulates the occurrence of security attacks in Wireless Sensor Networks (WSNs) by injecting events into real application simulators [14]. The framework NETA for simulation of communication network attacks based on INET has been presented in [7]. It relies on implementing *attacker nodes*, which can strike attacks when triggered at runtime through dedicated control messages. In [4], Queiroz *et al.* present SCADASim, a simulation tool to test the effect of attacks in SCADA systems.

Although it displays similarities with SenSec, NETA, and SCADASim, our simulation framework results more flexible and easier to use. In fact, SEA++ assumes that attacks have been successfully performed, and reproduces their effects on the network and application, rather than their actual performance. Also, SEA++ does not require the user to implement or customize any component of the simulation platform. This is particularly important for two reasons. First, it allows us to use off-the-shelf simulators. Second, good simulators are always the result of many man-years effort and therefore any modification is preferably avoided. In [6], we have presented ASF++, a framework akin to SEA++ but especially targeted to WSNs. A prototype implementation is available at [8]. In contrast, SEA++ provides a simulation framework for a conventional networking setting.

3 SEA++: Simulative Evalution of Attacks

The tool SEA++ is composed of the following three components. First, an *Attack Specification Language* (ASL) which allows the user to describe an attack to be evaluated, in terms of their practical and final *effects*. Second, an *Attack Specification Interpreter* (ASI) that converts the attack descriptions into configuration files for the attack simulation. Finally, an *Attack Simulator* that simulates the effects of specified attacks on the system under investigation, so making it possible to quantitatively evaluate their impact on the network and application.

Practically, the user first describes the effects of attacks to be evaluated by means of the high-level *Attack Specification Language*. Such descriptions can possibly be stored for later reuse. After that, the user runs the *Attack Specification Interpreter*, to convert the attack descriptions into an attack configuration file, which is provided as input to the *Attack Simulator*. Finally, the user runs the *Attack Simulator* and simulates the execution of the system affected by the described attacks. Note that the user is not required to further implement or customize any component of the *Attack Simulator*, with particular reference to application and communication modules.

3.1 The Attack Specification Language

The high-level *Attack Specification Language* allows users to describe attacks to be evaluated. It is worth noting that here we are not interested in how an attack

can actually be mounted and carried out. Such an issue attains to the *feasibility* of the attack, i.e. the likelihood of a given threat to occur. Feasibility is the other dimension of risk assessment and is not the focus of SEA++. Instead, we are interested in evaluating the *impact* of successful attacks, i.e. their resulting consequences on the system. Practically, we quantitatively evaluate the effects of successful attacks. To fix ideas, let us consider a deception attack such as message injection. Then, we are not interested in how the adversary can inject fake messages in the system or in reproducing the actual message injection. Instead, our goal consists in understanding and evaluating what are the final effects of such messages on the network and application, once they have been successfully injected.

From this standpoint, we assume that the successful execution of an attack produces a sequence of *events* that takes place atomically. The ASL consists in a collection of *primitives* that allow us to specify the sequence of events related to a given attack. Primitives are organized into two sets, as described below.

i) *Node primitives*, that account for physical attacks performed against nodes, and allow us to describe alterations in node behavior. In particular, the *node primitives* are:

- destroy(nodeID, t) removes node nodeID from the network at time t, preventing it from taking part in further communication.
- move(nodeID, pos, t) moves node nodeID to position pos at time t.

ii) *Message primitives*, that account for cyber attacks, and allow us to describe actions on network messages, including eavesdropping, altering, injection and dropping. In particular, the *message primitives* are:

- drop(pkt) discards the packet pkt.
- create(pkt, fld, content, ...) creates a new packet pkt and fill its field fld with content. A single invocation makes it possible to specify the content of multiple fields.
- 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.

The ASL provides additional statements that allow us to specify the occurrence of a list of events described through *message primitives*. For instance, the statement `from T every P do {<list of events>}` specifies that the list of events takes place periodically, with period P, on the declared list of nodes since time T.

Also, the ASL allows us to specify the *conditional* occurrence of events described through *message primitives*, depending on specific conditions evaluated by nodes at runtime. For instance, the following statements specify that the list

of events takes place on the declared list of nodes if condition is evaluated as TRUE.

```
from T nodes = <list of nodes> do {
  filter(<condition>) <list of events>
}
```

By means of the statements shown above, the ASL makes it possible to describe even complex attacks in a concise although clear way. For instance, let us consider a *wormhole attack* [11] starting at time 200 s, where node 9 tunnels MAC packets sent by node 5 to a remote area of the network containing nodes 10, 11 and 15. The attack can be described as follows:

```
dstList={10,11,15};
from 200 nodes = "9" do {
  filter(MAC.source==5 and MAC.type==DATA)
    put(packet,dstList,RX,0);
}
```

Note that 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. It follows that the user must be aware of the actual specific network protocols that are adopted at each communication layer. Also, for each of them, the user must be aware of the packet header structure and fields, and the specific capabilities possibly offered by the simulation platform. For instance, the OMNeT++ platform [2] and the INET framework [1] considered by SEA++ provide a set of objects, namely *descriptors*, which allow us to handle packets of a given communication layer and conveniently access their header fields.

3.2 The Attack Simulator

The *Attack Simulator* module considers every node as implemented by a *Enhanced Network Node* module. The latter is in turn composed of an *Application* module, a *Communication Stack* module, and a *Local Event Processor* (LEP) module. The *Application* module may be composed of different sub-modules modelling the actual node application. Similarly, the *Communication Stack* module may include an arbitrarily complex combination of protocols for different communication layers, e.g. transport, routing and MAC. All sub-modules but LEP can be off-the-shelf.

The LEP module is responsible for the management of events related to attacks, and operates transparently with respect to the other components of the *Enhanced Network Node* module. In particular, the LEP module intercepts all application and network packets traveling through a node's communication stack. Then, depending on the considered attacks to be evaluated, it can inspect and alter packets' content, inject new packets, or even discard intercepted ones. Finally, the LEP module can also alter the node behavior at different layers, change its position in space, or even neutralize the node by making it inactive.

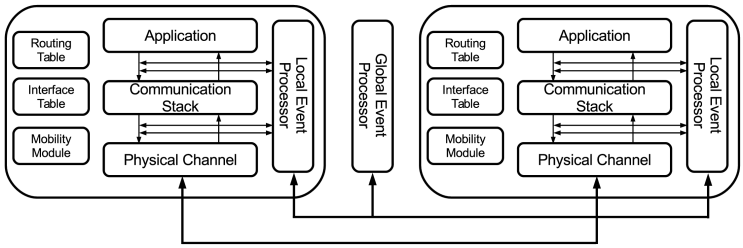


Fig 1. The Attack Simulator architecture

A system composed of multiple nodes is simulated by instantiating an *Enhanced Network Node* module for each node, and a single *Global Event Processor* (GEP) module that connects all the *Enhanced Network Node* modules with one another. In particular, the GEP module is separately connected with every LEP module, so allowing them to synchronize and communicate with one another in order to implement complex distributed attacks, such as a wormhole attack. Fig 1 depicts the architecture of the *Attack Simulator* component, with reference to a system composed of two interconnected nodes.

3.3 Prototype implementation for INET

We have implemented and released a prototype of SEA++. The *Attack Specification Interpreter* and *Attack Simulator* components are available at [9]. With reference to Fig 1, as to the *Application* and *Communication Stack* modules we used INET [1], an off-the-shelf simulator for wired, wireless and mobile networks, based on the discrete-event simulation platform *OMNeT++* [2].

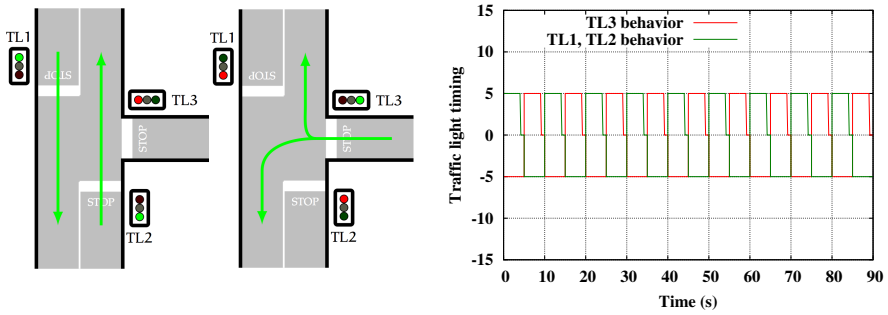
In the original INET architecture, network nodes are composed of different sub-modules. Also, nodes comprise a full communication stack composed by a transport, routing and MAC layer. INET provides the implementation of different communication protocols for each of such layers, as well as different network communication interfaces and physical channels. Thanks to the available communication stack, application running on the nodes can send/receive packets to/from the considered physical channel.

In our implementation of SEA++, we integrated the Local Event Processor and the Global Event Processor within the INET simulator. In particular, the Local Event Processor has been adapted to INET, in order to correctly manage simulation events and network packets. With reference to Fig 1, the Local Event Processor intercepts incoming and outgoing packets traveling through a node's Communication Stack, between every pair of layers.

4 Case study: a traffic light system

In this section, we consider a traffic light application scenario, and use SEA++ to evaluate the impact of two security attacks. In particular, we refer to the

T-intersection depicted in Fig 2, including a secondary one-way road that intersects a main road. The vehicular traffic in the intersection is managed by means of three traffic lights, i.e. TL1 and TL2 on the main road, and TL3 on the secondary road. We assume that a single *Traffic Controller* node periodically sends control messages to the three traffic lights (every 2.5 seconds in our setting), with the intent to adapt their behavior to the experienced vehicular traffic. Furthermore, the three traffic lights periodically send a feeedback message to the Traffic Controller (every 0.2 seconds in our setting), reporting about the actual traffic light time experienced during the last time interval. So doing, the Traffic Controller can check the correct behavior of traffic lights, and possibly adjust them by means of additional control messages. We assume that the regular traffic light timing has a period of 10 seconds, and is set as $\{5s; 1s; 4s\}$. That is, the red light is on for 5 seconds, followed by the yellow light active for 1 second, after which the green light is on for 4 seconds before concluding the period. This is shown in the graph reported in Fig 2, where values 5, 0 and -5 stand for green light on, yellow light on and red light on, respectively.



(a) Example of traffic flow **(b)** Regular traffic light timing

Fig 2. Traffic light scenario

Hereafter, we consider an adversary who has managed to compromise the traffic light TL3, so being able to drop and alter feedback messages intended to the Traffic Controller. Also, the considered adversary is able to perform a number of attacks against the network. For instance, she can inject fake control messages intended to TL3. Having said that, the final goal of the adversary consists in altering the behavior of TL3, in order to create inconsistent traffic light configurations, which can be dangerous or prone to traffic stalemate, when both directions have red and green light, respectively. Besides, the adversary is insterested in concealing the effects of performed attacks to the Traffic Controller.

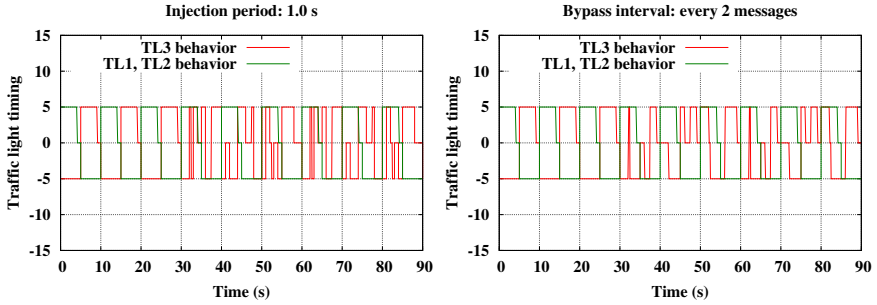
4.1 Attack impact and ranking

In the following, we consider two distinct attacks. In the first attack, namely *Injection*, the adversary regularly injects faked control messages intended to the traffic light TL3, specifying a traffic light timing $\{2s; 2s; 2s\}$. Then, upon

receiving a fake control message, TL3 sets the traffic light period to 6 seconds, and starts to observe a traffic light timing $\{2s; 2s; 2s\}$, i.e. 2 seconds are assigned to each one of the red, yellow and green light. We refer to different *injection periods*, i.e. different time intervals between two consecutive transmissions of fake control messages. Note that, upon receiving a genuine control message from the Traffic Controller, TL3 starts again the regular traffic light period of 10 seconds, according to the regular traffic light timing $\{5s; 1s; 4s\}$.

In the second attack, namely *Bypass*, TL3 ignores *some* genuine control messages received from the Traffic Controller. Specifically, TL3 may bypass some control messages upon their reception, and instead set the traffic light period to 6 seconds and start to observe a traffic light timing $\{2s; 2s; 2s\}$. That is, 2 seconds are assigned to each one of the red, yellow and green light. We refer to different *Bypass intervals*, i.e. the number of control messages before the next one to be bypassed. Note that, in case a control message is regularly accepted and processed, TL3 starts again the regular traffic light period of 10 seconds, according to the regular traffic light timing $\{5s; 1s; 4s\}$.

In both attacks, TL3 keeps on regularly sending feedback messages to the Traffic Controller, although always specifying the *expected* regular traffic light timing, i.e. $\{5s; 1s; 4s\}$. As a consequence, the Traffic Controller is not able to recognize that the observed traffic light timing differs from the expected one.



(a) Injection attack **(b)** Bypass attack

Fig 3. Traffic light timing under attack

Fig 3a and Fig 3b report the behavior of the traffic light system when the *Injection* attack or the *Bypass* attack are performed. Specifically, Fig 3a considers an *Injection period* of 1 s, whereas Fig 3b considers a *Bypass interval* of 2 messages. In both cases, the considered attack starts at time $t = 30$ s and is then performed throughout the simulation experiment. The two lines in the graphs depict the evolution of the traffic light configuration over time, separately for the main and the secondary road. Besides, every overlap of the two lines denote a misbehavior due to the considered attack, i.e. the occurrence of an undesired configuration.

Table 1 sorts the considered attacks according to their severity. In particular, attacks are sorted according to the amount of time that the system under attack experiences in a misbehavior state (see column *Incorrect*). Results are expressed

Table 1. Attack ranking

	Incorrect	**GG**	**GY**	**YY**	**RR**	**Correct**
Bypass (all messages)	51%	14%	17%	3%	17%	49%
Injection (Period 0.5 s)	41%	11%	14%	3%	13%	59%
Injection (Period 1 s)	36%	9%	12%	3%	12%	64%
Injection (Period 1.5 s)	35%	9%	9%	3%	14%	65%
Bypass (every 2 messages)	24%	8%	8%	0%	8%	76%
Bypass (every 3 messages)	9%	1%	6%	0%	2%	91%

as the percentage of time when the system observes a given traffic light configuration, while being under attack. In particular, the column *Incorrect* refers to all possible undesired configurations observed on the main and secondary road, i.e. Green-Green (GG), Green-Yellow (GY), Yellow-Yellow (YY) and Red-Red (RR). Separate results for each undesired configurations are reported in the relative dedicated columns. Finally, the *Correct* column refers to all possible licit configurations, i.e. Red-Green and Red-Yellow. We considered *Injection periods* 0.5 s, 1 s and 1.5 s for the *Injection* attack, and *Bypass intervals* 1, 2 and 3 messages for the *Bypass* attack.

As reported in Table 1, bypassing all control messages from the traffic controller results to be the most effective attack against the traffic light system. This suggests that this attack, especially when mounted at its maximum extent, is the one which deserves more to be addressed and counteracted. The *Injection* attack follows right after. In particular, as expected, the shorter the injection period, the more the attack is effective. Similarly, the *Bypass* attack is more effective when larger *Bypass intervals* are considered.

5 Conclusions

We have presented SEA++, a tool for simulative evaluation of attack impact based on the INET framework and the OMNeT++ platform. SEA++ allows the user to describe cyber-physical attacks and quantitatively evaluate their effects on the network and application. SEA++ does not require the user to modify any component of the simulation platform. As a case-study, we have showed the use of SEA++ to evaluate the impact of two different attacks on a traffic light management application scenario. In future work, we will integrate additional off-the-shelf simulators (e.g., Castalia, Simulink) to apply SEA++ to more complex systems such as smart grids and industrial plants. Finally, we intend to introduce the node primitive `disable` which complements `destroy` and disables every application activity of a node.

Acknowledgments

This work was supported by the EU FP7 Project SEGRID (Grant Agreement no. FP7-607109), and the PRIN Project TENACE (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

References

1. INET Framework - OMNeT++, `http://inet.omnetpp.org/`
2. OMNeT++ Network Simulation Framework, `http://www.omnetpp.org/`
3. B. Genge, C. Siaterlis and M. Hohenadel: AMICI: An Assessment Platform for Multi-domain Security Experimentation on Critical Infrastructures. In: Critical Information Infrastructures Security, Lecture Notes in Computer Science, vol. 7722, pp. 228–239. Springer Berlin Heidelberg (2013)
4. C. Queiroz, A. Mahmood and Z. Tari: SCADASim–A Framework for Building SCADA Simulations. IEEE Trans. on Smart Grid 2(4), 589–597 (December 2011)
5. C. Siaterlis, A.P. Garcia and B. Genge: On the Use of Emulab Testbeds for Scientifically Rigorous Experiments. IEEE Communications Surveys Tutorials 15(2), 929–942 (Second 2013)
6. G. Dini and M. Tiloca: ASF: an Attack Simulation Framework for wireless sensor networks. In: The 8th IEEE Intl. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob 2012). pp. 203–210 (October 2012)
7. L. Sánchez-Casado, R.A. Rodríguez-Gómez, R. Magán-Carrión and G. Maciá-Fernández: NETA: Evaluating the Effects of NETwork Attacks. MANETs as a Case Study. In: Advances in Security of Information and Communication Networks, Communications in Computer and Information Science, vol. 381, pp. 1–10. Springer Berlin Heidelberg (2013)
8. M. Tiloca, A. Pischedda, F. Racciatti, G. Dini: ASF++, An Attack Simulation Framework. https://github.com/asfpp (2015)
9. M. Tiloca, F. Racciatti, G. Dini: SEA++, a tool for Simulative Evaluation of Attacks. https://github.com/seapp (2015)
10. T. Bonaci, L. Bushnell and R. Poovendran: Node capture attacks in wireless sensor networks: A system theoretic approach. In: The 49th IEEE Conference on Decision and Control (CDC 2010). pp. 6765–6772 (December 2010)
11. Y.-C. Hu, A. Perrig and D.B. Johnson: Packet leashes: a defense against wormhole attacks in wireless networks. In: The Twenty-Second Annual Joint Conference of the IEEE Computer and Communications (INFOCOM 2003). vol. 3, pp. 1976–1986 (March 2003)
12. Y.-L. Huang, A. A. Cárdenas, S. Amin, Z.-S. Lin, H.-Yi Tsai and S. Sastry: Understanding the physical and economic consequences of attacks on control systems. International Journal of Critical Infrastructure Protection 2(3), 73–83 (2009)
13. Y. Mo, T. H.-J. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig and B. Sinopoli: Cyber-Physical Security of a Smart Grid Infrastructure. Proceedings of the IEEE 100(1), 195–209 (January 2012)
14. Y.-T. Wang and R. Bagrodia: SenSec: A Scalable and Accurate Framework for Wireless Sensor Network Security Evaluation. In: The 31st Intl. Conf. on Distributed Computing Systems Workshops (ICDCSW 2011). pp. 230–239 (June 2011)

15. Y. Xu, G. Chen, J. Ford and F. Makedon: Detecting Wormhole Attacks in Wireless Sensor Networks. In: Eric Goetz and Sujeet Shenoi (ed.) Critical Infrastructure Protection, Post-Proceedings of the First Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection. IFIP, vol. 253, pp. 267–279. Springer (March 2007)