A Connection Pattern-based Approach to Detect Network Traffic Anomalies in Critical Infrastructures

Béla Genge[1], Dorin Adrian Rusu[2], Piroska Haller[1]
[1]"Petru Maior" University of Tîrgu Mureş, Romania,
[2] VU University Amsterdam, The Netherlands
bela.genge@ing.upm.ro, d.rusu@student.vu.nl, phaller@upm.ro

ABSTRACT

Recent trends in Critical Infrastructures (CIs), e.g., power plants and energy smart grids, showed an increased use of commodity, off-the-shelf Information and Communication Technologies (ICT) hardware and software. Although this enabled the implementation of a broad palette of new features, the pervasive use of ICT, especially within the core of CIs, i.e., in Industrial Control Systems (ICSs), attracted a new class of attacks in which cyber disturbances propagate to the physical dimension of CIs. To ensure a more effective detection of cyber attacks against the ICS of CIs, we have developed SPEAR, a systematic approach that automatically configures anomaly detection engines to detect attacks that violate connection patterns specific to ICSs. The approach is validated by experimental scenarios including traffic traces from real industrial equipment and real malware (Stuxnet).

Categories and Subject Descriptors

J.7 [**Computers In Other Systems**]: Industrial control; C.2 [**Computer-Communication Networks**]: Security and protection

General Terms

Critical Infrastructures, Security

Keywords

Industrial Control Systems, Anomaly Detection Systems

1. INTRODUCTION

The term Critical Infrastructure (CI) underlines the significance of an infrastructure, which "if disrupted or destroyed, would have a serious impact on the health, safety, security or economic well-being of citizens" [10]. Although this general definition embraces installations from several industrial domains such as power generation & transmission, oil & gas industries, water & wastewater management, a common well-recognized factor amongst today's Critical Infrastructures is the adoption of commodity, off-the-shelf Information and Communication Technologies (ICT) hardware and software [3]. This particular trend is mainly a consequence of the advantages of pervasive ICT, which enabled the implementation of new services and features such as remote monitoring and maintenance, energy markets, and the newly emerging smart grid.

Nevertheless, this technological shift from a completely isolated environment to a "system of systems" integration had a dramatic impact on the security of CIs. By leveraging attack vectors that are commonly used to attack traditional computer systems, e.g., phishing and USB key infections, malware aimed at the disruption of critical infrastructure systems have become effective cyber weapons [6, 7]. Such attacks are usually targeted against the Industrial Control Systems (ICSs), which are part of the core of CIs.

With the continuing increase in the level of sophistication as well as in the number of yearly reported malware aimed at ICSs, nowadays, the development of protective measures to secure CIs is receiving considerable attention. As such, recent advancements in the field of CI security highlighted the applicability of anomaly detection techniques to efficiently detect abnormal behavior [14, 13, 16, 18]. In fact, anomaly-based intrusion detection is well-suited for scenarios in which the encountered behavior is sufficiently narrow to allow meaningful detection from the "normal". Therefore, in this paper we propose SPEAR, a systematic approach consisting of a tool-suite aimed at modeling the topology of ICSs and automatically generating Snort [17] detection rules. SPEAR relies on the predictive behavior of connections between different ICS hosts in order to identify abnormal packet exchanges. It builds on the assumption that the core of CIs, i.e., its ICSs, once deployed, remains fixed over long time periods, while eventual changes can have various causes, such as the replication, relocation or decommissioning of equipment [14, 5]. Nevertheless, the same assumption can be applied to communication flows between equipment, which exhibit long-lasting patterns, called *connection patterns* [15].

The mechanisms implemented within SPEAR highlight two phases: modeling of ICS networks and generating anomaly detection rules. In the first phase SPEAR provides a formal language based on ns-2 [1] and a graphical interface to model the architecture of ICSs as well as communication flows between equipment. The second phase provides an approach that processes the ICS model and generates Snort anomaly detection rules. In this paper we provide the mathematical

EuroSec'14, April 13-16 2014, Amsterdam, Netherlands
Copyright 2014 ACM 978-1-4503-2715-2/14/04 ...$15.00.
http://dx.doi.org/10.1145/2592791.2592792

background behind SPEAR's ability to model ICSs and to generate ADS rules. Subsequently, we provide implementation details and experimental results proving that SPEAR is more than a proof of concept and it is actually a tool-suite available for the ICS community as open-source software (`http://www.ibs.ro/~bela/conpat.html`).

The remainder of this paper is organized as follows: Section 2 provides a background on ICSs and summarizes related work emphasizing the contributions of SPEAR; Section 3 provides the mathematical background of SPEAR and describes implementation details; Section 4 includes a detailed analysis of SPEAR's ability to detect attacks in several settings; concluding remarks are presented in Section 5.

2. BACKGROUND AND RELATED WORK

Although their architecture may vary from one installation to another, the following components are known to be specific to ICSs [11]: (i) SCADA Servers, also known as Master Terminal Units (MTUs); (ii) Programmable Logical Controllers (PLCs) and Remote Terminal Units (RTUs); and (iii) Human Machine Interfaces (HMI).

From an operational point of view, PLCs receive data from sensors, elaborate a local actuation strategy, and send commands to the actuators. PLCs also provide the data received from sensors to MTUs and eventually execute the commands that they receive. HMIs interact with MTUs and typically implement a graphical interface used by human operators to interact with the physical installation.

Although anomaly detection is a well-established field of research, connection pattern-based ADSs have been only recently proposed for ICSs. In the remaining of this section we provide an overview of the main anomaly detection techniques for ICSs available in the literature. The presentation focuses on approaches addressing the cyber dimension of ICSs, and in particular the communication infrastructure. Nevertheless, we have found relevant research on ADSs embodying the physical dimension of ICSs as well [13], which we consider to be out of the scope of this paper.

The work of Garitano, *et al.* [12], explored the periodicity of network traffic from a real SCADA system to construct a network traffic prediction model. It was shown that industrial traffic exhibits a predictive behavior, which can be efficiently explored in the construction of ADSs.

Goldenberg and Wool [14] used deterministic finite automata theory to build a detailed model of Modbus/TCP industrial protocol. The model captures several details such as source and destination IP addresses, Modbus Master and Slave identifiers, message type, etc. The approach was validated against a real industrial system, where it proved to be highly sensitive, yet with a low false-positive rate.

Zhao, *et al.* [18], proposed dynamic time warping and adaptive fuzzy C means to detect anomalies in industrial traffic. The effectiveness of the approach was tested against data originating from a real steel plant.

In the work of Barbosa, *et al.* [4], periodic traffic bursts are associated to independent traffic flows and are carefully monitored. The approach includes a learning period to monitor and tune the parameters of the anomaly detection system. Short-Time Fourier Transform is used to construct a traffic spectogram, which is then used as ADS.

In their more recent work [5], Barbosa, *et al.*, proposed a similar approach to the one described in this paper. The approach identifies connection patterns between different hosts and it constructs a list consisting of source, destination, protocol and server port, for each traffic flow. The approach includes a learning period during which traffic flows amongst hosts are learned and are used in the detection phase. The main concern proved to be the duration of the learning period since certain connections do not happen regularly and a long learning period might lead to misconfiguration due to running attacks.

This review confirmed that the regularity exposed by industrial traffic and more specifically the availability of connection patterns provides solid building blocks in the construction of effective ADSs. Compared to state-of-the-art, SPEAR brings several contributions: (i) it provides a mathematical model for ICS networks; (ii) it provides the mathematical tools for generating detection rules; and (iii) it leverages well-established tools and detection engines, i.e., Snort, which makes it easily applicable to real installations. Although SPEAR relies on expert knowledge to model the ICS topology, this provides an attack-free "learning period" and an effective detection of cyber attacks even in compromised ICSs. Nevertheless, we realize the advantages of automated learning periods and from this perspective SPEAR can also be seen as an extension or improvement of existing approaches, where the manual modeling of ICSs could be combined with an automated learning procedure.

3. PROPOSED APPROACH: SPEAR

This section outlines the basic building blocks of the proposed approach, first covering a high level view of SPEAR, and then sketching the details for each step, including modeling of ICS networks and generating Snort detection rules.

3.1 Overview of SPEAR

Fundamentally, SPEAR is built around two phases: in *phase 1* the architecture of ICSs, including equipment, detection nodes, networks, and communication flows, are modeled; in *phase 2* detection rules (for Snort) are generated.

In order to accomplish phase 1 SPEAR provides a graphical user interface accompanied by a formal description language to describe the architecture of ICSs and communication flows between equipment. The language is based on an extension of the ns-2 scripting language and it provides a natural approach to describe network topologies, equipment, anomaly detection hosts, and communication flows.

In phase 2 SPEAR automatically generates detection rules specific to each ADS identified in phase 1. At the time of this writing SPEAR generates only Snort detection rules, since Snort is one of the most widely deployed detection engines available today. Nevertheless, other detection engines might be taken into account in future developments.

3.2 Phase 1: Modeling ICSs

We model the typical architecture of an ICS with a traditional graph model $G = (V, E)$, where V is the set of *vertices* and $E \subseteq V \times V$ is the set of pairs of vertices, known as *edges*. Each vertex models typical ICS nodes such as PLC, HMI, RTU, ADS as well as network components such as switch and modem. Edges denote typical connections between network components such as wired or wireless link.

Besides the network topology, SPEAR requires the definition of traffic flows amongst ICS hosts. The set of all node pairs including traffic flow between them is modeled as $T \subseteq V \times V \times \{\mathtt{tcp}, \mathtt{udp}\}$. The first element of each tuple

$t = (s, d, k), t \in T$ denotes the traffic source, i.e., s, the second element denotes the traffic destination, i.e., d, and the third element denotes the protocol, i.e., the kind of traffic k.

This model provides a basic description of the network topology and of the traffic flows required by SPEAR to generate rules that detect attacks violating connection patterns. Nevertheless, the model can be further extended with more expressive capabilities to enable the inclusion of additional details, such as the type of industrial protocol, e.g., Modbus.

3.3 Phase 2: Generating ADS rules

The ICS model from phase 1 is processed by phase 2 in order to automatically generate detection rules specific to each ADS. Since traffic flow amongst two nodes does not include information on intermediary nodes, phase 2 applies a breadth-first search (BFS) algorithm to find the path from source to destination. Then, it identifies each ADS along the path and generates Snort rules to whitelist allowed traffic.

The identification of ADSs is based on the following two assumptions: (i) if the ADS is connected to a switch found along the path, we consider that the ADS can monitor all packets passing through the switch by using typical switch features such as Switched Port Analyzer (SPAN); and (ii) if the ADS is directly connected to a firewall or software router found along the path we assume that the ADS can monitor all packets passing through such nodes.

Using the BFS algorithm we build $A \subseteq V$, denoting the set of all ADSs and we define $adspath(t)$ function to return all ADSs found on the path of a specific traffic flow. Then, for each ADS $a \in A$ we build the traffic set $F^a = \bigcup\{t | t \in T \text{ } and \text{ } a \in adspath(t)\}$.

Based on this set we now generate Snort detection rules for TCP and UDP protocols. The basic structure of a Snort rule includes two sides: the left, i.e., source, and the right, i.e., destination. For instance, the following Snort rule will generate an alarm message `ALERT!` for each TCP packet originating from any port of host 10.1.1.1 and having as destination any port of host 10.1.1.2:

```
alert tcp 10.1.1.1 any -> 10.1.1.2 any (msg: "ALERT!")
```

Although Snort rules can have a much more complex definition, in SPEAR we use this basic, but effective approach to define detection rules. Consequently, using the notation introduced until now a Snort rule can be formalized as a tuple $(k^*, s^*, d^*, p_s^*, p_d^*, m)$, where: $k^* \in \{\texttt{tcp}, \texttt{udp}\}^*$ denotes the type of protocol(s) to which the rule applies; s^* and d^* denote several source and destination IP addresses, respectively; p_s^* and p_d^* denote several source and destination port numbers, respectively; and m is the generated message. We use R^a to denote the set of all rules for a specific ADS $a \in A$ and we use $\Rightarrow$ as an operator to store rules in R^a.

Intuitively, most of the components of each tuple (i.e., k^*, s^* and d^*) can be identified from the basic ICS model introduced in the previous sub-section. However, in it's current version SPEAR does not distinguish between different port numbers and we use `anyp` to denote the set of all port numbers. For m we assume a constant message, which can be easily replaced to generate a more elaborate alarm.

For TCP we assume a bidirectional exchange of packets. This means that for each traffic flow defined in the ICS model, rules are generated for both source and destination hosts. Therefore, for each ADS $a \in A$ and for each host $v \in V \setminus A$ that is monitored by a we build the set of hosts that exchange TCP packets with v, denoted by $H_v^a = \bigcup\{v' | (v, v', \texttt{tcp}) \in F^a \text{ } or \text{ } (v', v, \texttt{tcp}) \in F^a\}$. Then, the following Snort rule is generated, which will issue alarms if v sends a packet to hosts outside H_v^a:

$$\text{Rule1: } (\{k\}, \{v\}, NOT(H_v^a), \texttt{anyp}, \texttt{anyp}, \texttt{ALERT!}) \Rightarrow R^a,$$

where $NOT(H_v^a)$ is a function returning the set of all hosts, i.e., all IP addresses, outside H_v^a. As shown in the remaining of this section, this rule also applies to UDP, and therefore $k \in \{\texttt{tcp}, \texttt{udp}\}$.

A particular case of TCP, which also applies to other protocols such as UDP, is when a specific host does not exchange any TCP packets. In such scenarios H_v^a is empty and the following two rules are generated to raise alerts for any TCP or UDP packet sent/received by v:

$$\text{Rule2: } (\{k\}, \{v\}, NOT(\{v\}), \texttt{anyp}, \texttt{anyp}, \texttt{ALERT!}) \Rightarrow R^a,$$
$$\text{Rule3: } (\{k\}, NOT(\{v\}), \{v\}, \texttt{anyp}, \texttt{anyp}, \texttt{ALERT!}) \Rightarrow R^a.$$

For UDP, the traffic flow can be unidirectional or bidirectional. The procedure for generating Snort rules is similar to the one described for TCP and it boils down to the construction of $H_v^a \subset V^*$. However, in this case $H_v^a = \bigcup\{v' | (v, v', \texttt{udp}) \in F^a\}$, which means that if bidirectional UDP traffic is modeled, a second tuple $(v', v, \texttt{udp}) \in F^a$ must be added. Therefore, in the bidirectional setting Rule1 is applied twice (once for each direction), while in the unidirectional setting both Rule1 and Rule2 are applied.

Finally, we must ensure that the two protocols which are governed by SPEAR, i.e., TCP and UDP, are running only on the hosts that have been modeled. Therefore, the following rule is generated for TCP and UDP as well:

$$\text{Rule4: } (\{k\}, \{NOT(V)\}, \{NOT(V)\}, \texttt{anyp}, \texttt{anyp}, \texttt{ALERT!}) \Rightarrow R^a.$$

3.4 Implementation Details

In order to provide an intuitive tool to model ICSs, in SPEAR we have adopted and extended the *Emulab Client GUI* [8] (ECG). ECG is a Java-based front-end to the Emulab testbed and was developed within the Emulab project.

The original ECG generates network topology descriptions in an extension of the ns-2 language [9] and provides an interface with the ability to model four entities: node, link, switch, and modem. Conversely, SPEAR brings several extensions to ECG to support typical equipment found in ICS. Thus, SPEAR provides additional graphical objects to model ICSs, while relying on the same mechanism to generate ns-2-based descriptions of network topologies.

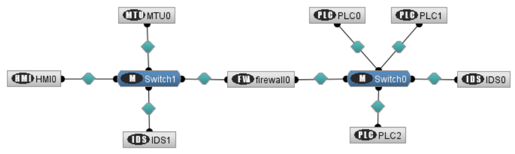

Figure 1: Example ICS architecture modeled with SPEAR's Emulab Client GUI.

An example ICS modeled with ECG is presented in Figure 1. The topology includes two networks: (i) the control network comprising of three PLCs and one ADS; and (ii) the process network comprising of one HMI, one MTU and one ADS. In this example we have defined TCP traffic between MTU0 and three PLCs and between MTU0 and HMI0. The following ns-2 script (partially shown) was generated:

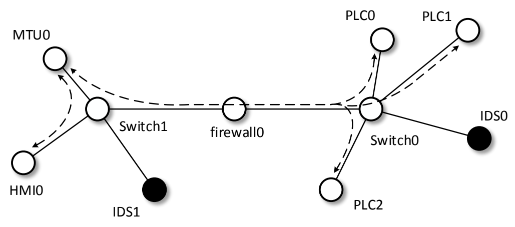

Figure 2: Example ICS graph representation.

```
set ns [new Simulator]
# Nodes
set MTU0 [$ns node]
tb-set-node-os $MTU0 ncSCADA-MTU
# Lans
set Switch0 [$ns make-lan "$firewall0 $IDS0 $PLC0 ..."]
# Event Agents
set tg0 [new Application/Traffic/CBR]
```

As illustrated by the previous listing, the generated ICS description is mostly based on the ns-2 language. Commands starting with `tb-`, i.e., testbed, have been defined for the Emulab project in order to express specific requirements of emulation testbeds. Once the ICS topology is modeled by SPEAR's ECG, it is forwarded to a Python-based script to automatically generate Snort rules. As shown in Figure 2, the graph constructed in phase 2 has a similar structure to the original network topology. Thus, phase 2 identifies two detection nodes, depicted as IDS0 and IDS1, and builds T, A, and F. Then, it generates Snort detection rules for each of the two ADSs. To illustrate the rules generated by SPEAR, we provide the following listing for IDS1:

```
ipvar $MTU0 [10.1.1.2]
...
1. alert tcp $MTU0 any -> ![$PLC0,$PLC1,$PLC2,$HMI0] any (...)
2. alert tcp ![$PLC0,$PLC1,$PLC2,$HMI0] any -> $MTU0 any (...)
3. alert tcp $HMI0 any -> !$MTU0 any (...)
4. alert tcp !$MTU0 any -> $HMI0 any (...)
5. alert tcp $firewall0 any -> !$firewall0 any (...)
6. alert tcp !$firewall0 any -> $firewall0 any (...)
7. alert tcp ![$MTU0 $PLC0 ...] any -> ![$MTU0 $PLC0 ...] any (...)
8. alert udp any any -> any any (...)
```

As depicted in the above listing, the first two rules were generated by Rule1 in order to raise alerts if MTU0 exchanges TCP packets with nodes other than PLC0, PLC1, PLC2, and HMI0. The third and fourth rules were generated with the same rule in order to raise alarms if HMI0 exchanges packets with nodes other than MTU0.

The next two rules raise alarms if firewall0 sends/receives any TCP packets. These were generated by Rule2 and Rule3 since we did not define any TCP traffic between firewall0 and other nodes. The last two rules will raise alarms on any TCP packet that was not issued by a modeled host and on any UDP packet, since UDP traffic was not defined for this ICS. The two rules were generated by Rule4.

4. EXPERIMENTAL ASSESSMENT

This section summarizes the results concerning the assessment of SPEAR from several perspectives. First, we illustrate SPEAR's ability to detect attacks in traffic traces captured from a laboratory installation including real equipment and real malware. Then, we evaluate its ability to detect attacks in traffic traces generated by simulations. Finally, we measure the execution time of SPEAR's rule generator script in order to evaluate its scalability.

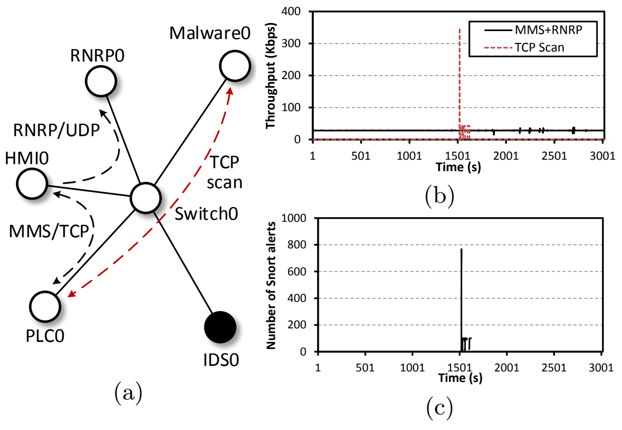

Figure 3: Experimental setting consisting of real industrial equipment: (a) ICS topology; (b) network traffic; and (c) alarms raised by Snort.

4.1 Detecting Attacks in Real Industrial Traces

We have set-up an experiment consisting of a PLC and HMI software from ABB. On the HMI node we hosted MTU software from ABB to ensure communication with PLC through Manufacturing Message Specification (MMS) protocol over TCP. During the experiment we noticed that the HMI also sends Redundant Network Routing Protocol (RNRP) packets over UDP to a specific RNRP router. Although the router was not present in this experiment, the feature remained active from previous configurations. Consequently, in order to reflect the correct behavior of HMI, we have modeled an RNRP router and we defined unidirectional UDP traffic between HMI and RNRP router. Finally, we added a generic host to the experiment to denote a compromised, malicious node, on which we ran *nmap* software. Then, we launched several TCP scans against the PLC, including TCP-SYN, TCP-NULL, TCP-FIN, and TCP-XMAS scans. The experiment topology is illustrated in Figure 3 (a).

In order to test SPEAR's ability to detect the attack, we modeled this topology using SPEAR's ECG software and we have generated Snort rules with SPEAR's rule generator. Then, we have added a detection node to the experiment and we have configured Snort to run the generated rules.

The network traffic throughput including MMS and RNRP are illustrated in Figure 3 (b). Here we have also depicted the traffic generated by *nmap*, where the first, highly visible burst is due to the TCP-SYN scan. This is then followed by smaller traffic bursts depicting the throughput of TCP-NULL, TCP-FIN, and TCP-XMAS scans.

As shown in Figure 3 (c), with the rules generated by SPEAR, Snort is able to accurately detect the implemented attack. Since the attack violates the connection patterns defined for this topology, Snort generates an alert message for each packet that matches a specific rule.

Although the attack is already visible from the network traffic throughput depicted in Figure 3 (b), we underline the fact that by using connection patterns, detection engines can be highly efficient and can detect attacks relying even on one single packet. However, an attacker can still exploit connection patterns defined for each compromised host. For instance, in the scenario beforehand if the attacker compromises HMI0, then he/she can send UDP packets to RNRP0

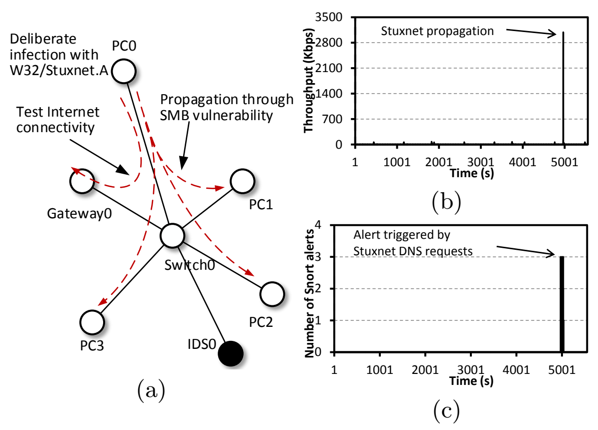

Figure 4: Experimental setting including infection with Stuxnet malware: (a) ICS topology; (b) network traffic; and (c) alarms raised by Snort.

and TCP packets to PLC0, which will not be detected. Nevertheless, typical malware (also the case of Stuxnet [6]) will try to spread, to compromise other stations and to initiate connections with other hosts as well. In such cases SPEAR will issue several alarms, even in the presence of low-rate attacks, which might not be detected by other approaches.

4.2 Detecting Attacks Involving Real Malware

The main concern regarding the previous experiment is that the simplicity of the synthetic attack implemented through *nmap* might exhibit significant deviations from the behavior of real malware. In fact, real malware might exploit regular network traffic to hide its presence. Therefore, we have tested SPEAR and its ability to detect the presence of real malware in a network setting involving infected computers.

For this experiment we configured a network with four hosts running Windows XP SP2 and we deliberately infected one of the hosts with Stuxnet malware (`W32/Stuxnet.A`). We monitored the network traffic and we verified that Stuxnet "installed" itself successfully on the infected host. Although the experiment did not include software or control hardware from Siemens, network traces showed that Stuxnet exploited vulnerability MS08-067 within SMB protocol (used for sharing files and other resources between computers) and copied itself over the network to the other hosts. Additionally, Stuxnet tested Internet connectivity by trying to connect to `www.windowsupdate.com` and `www.msn.com`. The experiment setting is illustrated in Figure 4 (a).

Next, we used SPEAR to construct the ICS model and we defined regular TCP traffic between the four hosts (MS Windows SMB traffic). Since the Windows hosts did not require a DNS server, we did not model such traffic.

The exact moment when Stuxnet copies itself onto another host is clearly visible in Figure 4 (b). Since Stuxnet exploits vulnerabilities in SMB protocol, with the rules generated by SPEAR, Snort does not raise alarms when Stuxnet propagates to other hosts. Nevertheless, Snort raises alarms for each DNS request issued by Stuxnet, since this traffic was not defined in the network model (Figure 4 (c)). Although not visible in these figures, similar DNS queries are sent from the moment the first host was infected. This underlines once again the effectiveness of the proposed approach which can trigger alarms on a per-packet basis.

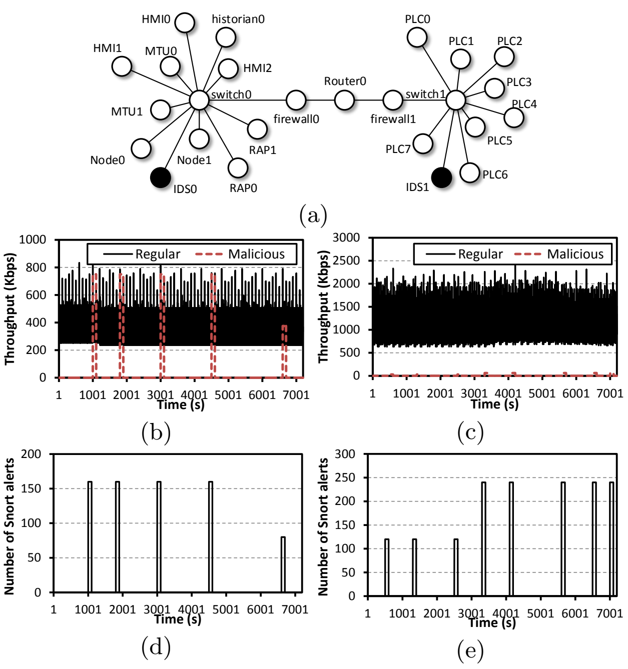

Figure 5: Experimental setting consisting of a simulated topology: (a) ICS topology; (b) and (c) regular and malware traffic throughput in setting 1 and 2, respectively; and (d) and (e) the number of alarms raised by Snort in setting 1 and 2, respectively.

4.3 Detecting Attacks in Simulated Traces

In order to test the efficiency of SPEAR with larger topologies we have recreated in a simulated environment a typical ICS topology including two networks: control network and process network. For each network we have defined around 10 nodes, regular UDP traffic between several nodes and between the two networks. Subsequently, we have defined malicious UDP traffic between eight nodes. Then, we used SPEAR's ability to generate Snort rules. The topology used in this scenario is shown in Figure 5 (a). In order to ensure clarity of presentation, traffic lines (regular and malicious) were excluded from this figure.

Within this scenario we also illustrate the effectiveness of connection patterns in the presence of low-rate attacks. As such, we defined two settings: in setting 1 the throughput of the attack is configured to a similar rate as the regular traffic, while in setting 2 the throughput of the attack is around 30 times lower than the average throughput of regular traffic.

In order to test Snort's ability to detect attacks in both settings, we recreated this ICS topology with the recent Network Simulator version 3, i.e., *ns-3* [2]. Then, we generated packet capture files and we ran Snort with the rules generated by SPEAR. As shown in Figure 5 (b), in setting 1 the attack is clearly visible. Its throughput reaches almost 800Kbps, which is similar to traffic peaks exhibited by regular traffic. For each attack packet that is detected, Snort issues an alarm (see Figure 5 (d)). On the other hand, the attack in setting 2 is less obvious. As shown in Figure 5 (c), the throughput of regular traffic reaches almost 2500Kbps, while the maximum attack throughput is 60Kbps. Nevertheless, Snort is able to efficiently detect irregular traffic, i.e., malicious, and raises alarms accordingly (Figure 5 (e)).

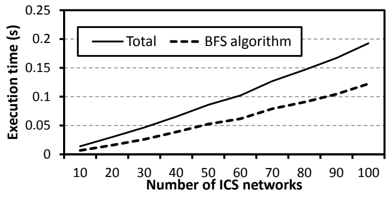

Figure 6: Execution time of rule generator.

4.4 Execution Time of Rule Generator Script

We aim to determine the execution time of the rule generator script with large ICS topologies. Therefore, we generated a total of 10 different ICS topologies, starting with a topology of 10 networks and gradually increasing the number of networks for each topology by 10. Each network included 10 hosts and one ADS; the largest topology included 100 networks with a total of 1000 hosts and 100 ADSs. Traffic generators were configured between all components of each network and between all networks.

As shown in Figure 6, the execution time of the implemented rule generator (in Python) exhibits a linear trend. For instance, the script executes under 20ms for 10 networks and under 200ms for 100 networks. For each topology the calculation of the shortest path within the graph (BFS) takes the longest time to execute. Overall, the results are a clear demonstration of the applicability of the script to large-scale infrastructures. Since the execution takes under 200ms even for a topology including 1000 hosts, we can consider that additional extensions, e.g., complex rules, are also feasible.

5. CONCLUSIONS

As we have shown, the definition of connection patterns between different hosts in the core of CIs, i.e., ICSs, provides an effective approach for implementing ADSs. The learning phase present in other approaches, which might lead to undetected attacks, is replaced by expert knowledge and a formal language for describing ICS topology. The ICS model is then used to automatically generate a set of detection rules for each ADS. Detection rules are specifically generated for Snort, which lead to the detection of attacks that violate the modeled connection patterns. The experiments from Section 4 demonstrated the applicability of SPEAR in real scenarios and on large ICS topologies.

The main advantages that SPEAR brings over state-of-the-art is that it automatizes the rule generation procedure for ICSs and a well-established detection engine, i.e., Snort, by employing available open-source tools. Nevertheless, the authors are aware of SPEAR's limitations as well. In fact, we intend to expand the set of supported protocols in order to provide more expressive modeling capabilities for industrial protocols. Furthermore, we realize the benefits of automated topology learning techniques, which are also planned to be integrated in SPEAR. Nevertheless, as shown in [5], automated learning procedures must be carefully planned since these might accidentally validate malicious traffic.

6. ACKNOWLEDGMENTS

This research was supported, in part, by a Marie Curie FP7 Integration Grant within the 7th European Union Framework Programme.

7. REFERENCES

[1] ns-2 network simulator. http://www.isi.edu/nsnam/ns/, 2006.
[2] ns-3 network simulator. https://www.nsnam.org/, 2014.
[3] R. Anderson and R. Hundley. The implications of COTS vulnerabilities for the DoD and critical U.S. infrastructures. *RAND Report*, P-8031, 1998.
[4] R. Barbosa, R. Sadre, and A. Pras. Towards periodicity based anomaly detection in SCADA networks. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation*, pages 1–4, September 2012.
[5] R. Barbosa, R. Sadre, and A. Pras. Flow whitelisting in SCADA networks. *International Journal of Critical Infrastructure Protection*, 6(3-4):150–158, 2013.
[6] T. Chen and S. Abu-Nimeh. Lessons from Stuxnet. *Computer*, 44(4):91–93, April 2011.
[7] CrySiS Lab. Flame - a complex malware for targeted attacks, May 2012.
[8] Emulab project. Emulab client GUI. https://www.emulab.net/netlab/client.php3, November 2005.
[9] Emulab project. NS command extensions. https://wiki.emulab.net/wiki/nscommands, 2014.
[10] European Commission. Communication from the Commission to the Council - Critical Infrastructure Protection in the fight against terrorism. *COM(2004)0702.*, October 2004.
[11] B. Galloway and G. Hancke. Introduction to industrial control networks. *Communications Surveys Tutorials, IEEE*, 15(2):860–880, Second 2013.
[12] I. Garitano, C. Siaterlis, B. Genge, R. Uribeetxeberria, and U. Zurutuza. A method to construct network traffic models for process control systems. In *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on*, pages 1–8, Sept 2012.
[13] B. Genge, C. Siaterlis, and G. Karopoulos. Data fusion-based anomay detection in networked critical infrastructures. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pages 1–8, June 2013.
[14] N. Goldenberg and A. Wool. Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75, 2013.
[15] E. Pleijsier. Towards anomaly detection in SCADA networks using connection patters. In *18th Twente Student Conference on IT*, pages 1–6, January 2013.
[16] F. Schuster, A. Paul, and H. Konig. Towards learning normality for anomaly detection in industrial control networks. In *Emerging Management Mechanisms for the Future Internet*, volume 7943 of *Lecture Notes in Computer Science*, pages 61–72. Springer Berlin Heidelberg, 2013.
[17] Sourcefire. Snort. http://www.snort.org/, 2014.
[18] J. Zhao, K. Liu, W. Wang, and Y. Liu. Adaptive fuzzy clustering based anomaly data detection in energy system of steel industry. *Information Sciences*, 259(0):335–345, 2014.