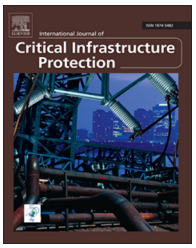
INTERNATIONAL JOURNAL OF CRITICAL INFRASTRUCTURE PROTECTION ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Available online at www.sciencedirect.com

ScienceDirect

www.elsevier.com/locate/ijcip



Exploiting traffic periodicity in industrial control networks

Rafael Ramos Regis Barbosa[a,c,*], *Ramin Sadre*[a,c], *Aiko Pras*[b,c]

[a]3D Hubs, Herengracht 182, 1016BR Amsterdam, The Netherlands
[b]INGI Computer Science Department, Catholic University of Louvain, Place Sainte Barbe 2, B-1348 Louvain-la-Neuve, Belgium
[c]Design and Analysis of Communication Systems, Faculty of Electrical Engineering, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

ARTICLE INFO

Article history:
Received 8 June 2015
Received in revised form
16 January 2016
Accepted 17 February 2016

Keywords:
Industrial control networks
Traffic periodicity
Traffic models
Intrusion detection
Device performance evaluation

ABSTRACT

Industrial control systems play a major role in the operation of critical infrastructure assets. Due to the polling mechanisms typically used to retrieve data from field devices, industrial control network traffic exhibits strong periodic patterns. This paper presents a novel approach that uses message repetition and timing information to automatically learn traffic models that capture the periodic patterns. The feasibility of the approach is demonstrated using three traffic traces collected from real-world industrial networks. Two practical applications for the learned models are presented. The first is their use in intrusion detection systems; the learned models represent whitelists of valid commands and the frequencies at which they are sent; thus, the models may be used to detect data injection and denial-of-service attacks. The second application is to generate synthetic traffic traces, which can be used to test intrusion detection systems and evaluate the performance of industrial control devices.

1. Introduction

Industrial control system (ICS) networks are commonly deployed to support the operation of critical infrastructure assets. Historically, industrial control networks have incorporated special-purpose embedded devices that communicate using proprietary protocols. During the past decade, however, the trend has been to adopt the TCP/IP protocol stack and use commercial off-the-shelf devices.

Despite the use of TCP/IP, communications in industrial control networks differ significantly from communications in the Internet or office local-area networks. In an industrial control network, data is continuously retrieved from field devices such as programmable logic controllers (PLCs), so that a real-time view of the supported industrial processes can be established. Typically, data is retrieved via automated, periodic polling processes run by industrial control devices (i.e., master devices). Manual interventions, such as sending commands to field devices, are rare. As a consequence, traffic regularities arise – the set of communicating nodes is stable and network traffic exhibits strong periodic patterns [3]. In fact, well-known characteristics of Internet traffic [12], such as self-similarity and heavy-tailedness, are generally absent [4].

This paper presents a novel approach for learning models of industrial control network traffic; the approach is

*Corresponding author at: 3D Hubs, Herengracht 182, 1016BR Amsterdam, The Netherlands.
E-mail address: rafael@3dhubs.com (R.R.R. Barbosa).

http://dx.doi.org/10.1016/j.ijcip.2016.02.004
1874-5482/

automated in a tool named PeriodAnalyser. Traffic models of industrial control networks are important in several application areas. For example, the trend toward TCP/IP-based protocols has made industrial control networks more vulnerable to the same threats that plague traditional information technology networks [11].

Due to the presence of large numbers of zero-day attacks, anomaly-based intrusion detection in industrial control networks is of particular interest [8]. In information technology networks, anomaly-based approaches typically exhibit high false-positive error rates due to the enormous variability of network traffic [22]. In contrast, the regularities present in industrial control network traffic make anomaly detection very promising. The models learned by PeriodAnalyser can be viewed as whitelists of valid commands and the frequencies at which they are sent. These whitelists provide protection against a number of attacks, including data injection and denial-of-service attacks.

Traffic models are also needed to generate synthetic traffic traces. Such traces, combined with attack traffic, can be used as the ground truth for testing and evaluating intrusion detection systems. Alternatively, the synthetic traces could be used to evaluate the performance of industrial control devices.

An extensive search of the literature reveals that the automated approach presented in this paper is the first to *directly* exploit periodicity in industrial control network traffic. Message repetition and timing information are used to detect periodic cycles in the traffic; this addresses the limitations of existing approaches. The approach is validated using traffic traces collected from operational industrial control networks – two water treatment plant networks and one electric-gas utility network.

2. Related work

The problem of periodic traffic pattern identification is different from the classical problem of cycle detection, which is solved by Floyd's algorithm [19]. The classical cycle detection problem involves (efficiently) detecting that a (large) sequence has become periodic under the assumption that the sequence is perfectly periodic. In contrast, the problem considered in this paper is similar to the problem of periodicity detection in temporal data studied by the data mining community [20]. In the data mining context, the problem involves finding repeating patterns in a time series of symbols from a certain alphabet, typically represented as a string (e.g., `abcdabyzabfg`). The approach proposed in [16] allows for imperfect patterns (i.e., patterns that do not reoccur in every cycle) and partial patterns (i.e., only a subset of the patterns are periodic). For example, the substring `ab` is considered to be periodic in the string `abcdabyzabfg`.

The difficulty in applying such methods to network traffic is to define a proper time bin size (i.e., interval between two symbols) in order to create the time series. If the chosen time bin size is too large, unrelated network messages end up in the same time bin and, hence, are represented by only one symbol; this obfuscates the periodic pattern. A very small time bin size could be used to reduce the extent of this problem, but not completely, because TCP can merge multiple application protocol data units (PDUs) into a single segment, causing the protocol data units to be observed at identical times. In addition, this solution is not efficient because it produces many empty time bins and long symbol strings.

Spectral analysis is commonly used to uncover periodicity in network traffic (see, e.g., [1,6,14]). Van Splunder [24] proposes an approach based on the variance of packet inter-arrival times to detect network traffic periodicity. In previous work [5], the authors of this paper investigated the idea of detecting anomalies in the periodic behavior of industrial control network traffic using tools such as discrete Fourier transforms and autocorrelation functions. The main limitation of these methods is the so-called "semantic gap" due to the fact that they operate on information based on the observed network packets (e.g., number of packets or bytes sent per time interval). While it is relatively easy to detect periodic activities using this information, little insight is provided into which packets caused the periodic behavior [16].

The work by Goldenberg and Wool [13] is closely related to the research presented in this paper. Based on the observation that traffic exchanged between a human–machine interface (HMI) and a programmable logic controller consists of requests for the same values being sent periodically, Goldenberg and Wool attempted to model Modbus traffic using a deterministic finite automaton (DFA). The automaton captures the order in which requests and their respective responses are normally exchanged and triggers alarms when an unexpected transition (i.e., unexpected sequence of two messages) is observed. The approach is able to automatically learn an automaton from a training set. However, because only a single automaton is used per connection, a connection carrying requests sent with different periods results in a large model. Moreover, small fluctuations that change the relative order of messages are not captured by the model. Goldenberg and Wool propose a two-level approach that reduces (but does not eliminate) the problem. A second limitation is that the model of Goldenberg and Wool captures the order of messages but not their inter-arrival times. For instance, the model cannot distinguish when a sequence of requests is sent every ten minutes and when the same requests are sent every ten milliseconds.

More recently, Caselli et al. [7] proposed a general approach to detect sequence attacks – sequences of "valid" events (e.g., network messages, log entries and variable values) that have an adverse impact on a system. Their sequence-aware intrusion detection system models normal behavior using discrete-time Markov chains. Anomalies are detected when unknown states are reached or unlikely or unknown transitions occur. Although periodic behavior can be indirectly captured in terms of sequences of states, complex periodic patterns (e.g., multiple periods with possible drift because of timing variations), which are addressed in the present work, would either result in a large model or a compact model that only provides a fuzzy description of the real process.

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

3. Period analysis

This section describes a novel approach for learning periodic traffic patterns and the implementation of the approach in the PeriodAnalyser tool. Although the approach does not rely on deterministic finite automata, as in the case of the Goldenberg–Wool methodology [13], communications to and from programmable logic controllers are modeled as a series of requests (and their responses).

PeriodAnalyser is composed of three modules: (i) Multiplexer; (ii) Tokenizer; and (iii) Learner. The Multiplexer module preprocesses the monitored network traffic and separates it into different flows. Next, the Tokenizer module transforms each packet into a protocol-independent format called a token. Finally, the Learner module processes each token to identify and characterize periodic activities called cycles. The information embodied in all the identified cycles constitutes the model of industrial control network communications.

It is important to acknowledge that the information captured by the Learner module might already be known (partially) to industrial control system operators or, alternatively, it could be extracted from the configuration files [15]. Consequently, this information could be used directly instead of learning it from the network traffic, thereby eliminating the need for the Learner module. However, experience with real-world critical infrastructure assets has shown that this information is not readily available or it is incomplete at best.

3.1. Requirements

The following four requirements have influenced the design of PeriodAnalyser:

- *Multiple cycles*: Multiple activities with different periods can occur in the same network connection. For instance, suppose that requests R_1, R_2, R_3 and R_4 are sent by a master device to a field device every 1 s, 1 s, 1 s and 2 s, respectively. Then, the proposed approach should identify two cycles: (i) one cycle with a period of 1 s consisting of the requests R_1, R_2 and R_3; and (ii) one cycle with a period of 2 s containing only the request R_4.
- *Periodicity at different aggregation levels*: Periodic patterns can be observed on "short-lived" or "long-lived" network connections. An industrial control system application may establish a new TCP connection for each cycle of the periodic message exchange, resulting in many short-lived connections. Alternatively, it may establish one long-lived connection that persists for several minutes or hours.
- *Request reordering*: The order of requests in a cycle iteration is not necessarily fixed. For example, iterations R_1, R_2, R_3 and R_1, R_3, R_2 should be considered equal. The lack of order can be caused, for instance, by a multithreading implementation of the application issuing the requests, where multiple threads compete for access to the network interface.
- *Timing variations*: Small variations in cycle timing should be expected. Delays and jitter can be introduced by the network [25] or by the server that generates the requests. For example, an application can be preempted if the load on the server is high. Aside from delays, packets can be lost, either by the network or by the measurement equipment.

3.2. Multiplexer module

The Multiplexer module filters industrial control protocol packets, discards the non-control-system traffic and multiplexes the packets onto different flows. This work considers two common industrial control protocols: Modbus/TCP and MMS [18,21]. The filtering is performed by identifying all the packets that contain Modbus or MMS application headers. Both protocols use TCP as the transport layer protocol. Packets that do not carry application data, such as TCP ACK packets or TCP handshaking packets, are discarded.

Two types of TCP connections were observed in the experimental datasets: (i) long-lived connections containing requests sent at periodic intervals; and (ii) short-lived connections that were established (and torn down) at periodic intervals. After filtering, packets belonging to the first type were grouped to a flow using the five-tuple: (Server Address, IP Protocol, Server Port, Client Address, Client Port) as the grouping key. The second type of packets were grouped using the four-tuple: (Server Address, IP Protocol, Server Port, Client Address) to aggregate the connections in a single flow. Short connections were detected based on their durations. In the experimental datasets, short connections typically had durations below one second.

3.3. Tokenizer module

The Tokenizer module transforms the packets that were filtered and grouped by the Multiplexer module to a common protocol-independent format. The resulting packets are passed to the Learner module. Note that only the Tokenizer and Multiplexer modules need to be adapted in order to accommodate new industrial control protocols.

For each observed packet, the Tokenizer module outputs a tuple comprising its timestamp and a token. If the packet has a request message, the token consists of two parts: (i) message identifier, which identifies the request; and (ii) pair identifier, which pairs the request and response messages. If the packet contains a response message, the token consists only of the pair identifier. Note that, whereas the contents of request messages (e.g., "get pump pressure") are expected to be repeated periodically, the contents of response messages (e.g., pressure value) are likely to vary considerably.

The token is based on application header fields; therefore, its generation is protocol dependent. Table 1 lists the protocol fields extracted from Modbus and MMS protocol data units to generate the message identifier and pair identifier that form a token.

3.4. Learner module

The Learner module processes the output of the Tokenizer module flow-by-flow with the goal of identifying all the cycles in each flow. To accomplish this, the sequence of requests in a flow is split into candidate iterations. If N identical candidate iterations are observed, then the learning algorithm

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

Table 1 – Mapping of Modbus and MMS protocol data units to tokens.

Protocol	PDU	Message Identifier	Pair Identifier
Modbus	Request	*length field, unit identifier, function code, data*	*transaction ID*
	Response	–	*transaction ID*
MMS	Confirmed request	*confirmedServiceRequest* and optional fields	*invokeID*
	Confirmed response	–	*invokeID*
	Unconfirmed request	*unconfirmedService* and optional fields	None
	Initiate request	"initiate"	"initiate"
	Initiate response	–	"initiate"
	Conclude request	"conclude"	"conclude"
	Conclude response	–	"conclude"

concludes that there is enough evidence for a cycle. Unlike the approach described in [13], it is not necessary to send requests in a fixed sequence to identify candidates. Instead, it is assumed that repeated requests delimit candidates – once a repeated request is observed, a new candidate cycle begins. This means that the same request does not appear in different cycles. Issuing the same request in different cycles (i.e., with different frequencies) is inefficient and should not occur in a real industrial control network.

To illustrate the approach, consider the sequence `abcdcdbadcbaabcd` of periodic requests `a`, `b`, `c` and `d`. The second appearance of `c` marks the beginning of a new candidate cycle. As a result, four candidate cycles are obtained, `abcd`, `cdba`, `dcba` and `abcd`, each containing all four requests. If the network traffic were to be captured at a later moment, say at the first `d`, then the first candidate would be `dc`, the second candidate would be `dba` and the third and fourth candidates would be `dcba` and `abcd`, respectively. Following the definition, only the last two candidates would be considered as identical (and identified as a cycle if $N=2$), while the first two candidates would be ignored. This demonstrates that the algorithm is able to synchronize itself to a cycle.

Having identified a group of N identical candidate cycles, it is necessary to verify if they demonstrate regular timing behavior. This is accomplished by calculating the duration of each candidate cycle. The duration of a candidate cycle is defined as the time between its first request and the first request of the following candidate cycle. If the difference between the minimum and maximum duration of a candidate cycle is above a threshold dur_{thr}, the candidate cycle is rejected because the grouping of the requests may have occurred by chance. Experiments revealed that $dur_{thr} = 1$ s worked well. Such delay variations should not occur in typical healthy industrial control networks even with highly fluctuating network latencies.

A preprocessing step is introduced to handle multiple cycles with different periods and non-periodic messages more efficiently. The preprocessing step groups requests according to their number of occurrences. The idea is that requests sent with the same frequency occur roughly the same number of times in the training data. In case multiple cycles are present in a flow, their requests should have different numbers of occurrences and, thus, should be separated during the preprocessing step. Similarly, non-periodic requests are likely to have different numbers of occurrences from periodic requests. Note, however, that even messages in the same cycle may have different numbers of occurrences (e.g., if the duration T of the training set is not a multiple of the cycle duration or if requests are lost).

Algorithm 1. Learner module algorithm.

Input: A tokenized flow and parameters N, ϵ, dur_{thr}
Output: A list of tuples (request set, $dur_{min}, dur_{max}, dur_{std}$)
/* `Step 1: group requests` */
count request occurrences;
  group requests with same counter $\pm\epsilon$;
  for *each group* **do**
    for *each subset in group* **do**
      /* `Step 2: find candidates` */
      for *each request in subset* **do**
        candidates ← N repeating cycles;
          /* `Step 3: test candidates` */
        dur_{min}, dur_{max} ← minimum and maximum candidate durations;
          if $dur_{max} - dur_{min} < dur_{thr}$
            dur_{std} ← cycle duration standard deviation;
            store (request set, $dur_{min}, dur_{max}, dur_{std}$);
            continue to next subset;
          else
            reset candidates;
        end
        ignore remaining subset requests as non-periodic;
  end
end

A tolerance ϵ is specified to deal with this issue. Requests with approximately the same number of occurrences ($\pm\epsilon$) are assumed to belong to the same group. In the experiments, ϵ was set to one.

Algorithm 1 shows the pseudo-code for the Learner module. The algorithm can be split into three steps:

1. *Group requests*: Count the number of occurrences of each request and create groups of requests with the same numbers of occurrences $\pm\epsilon$.
2. *Find candidate cycles*: Search each group for candidate cycles. A candidate cycle starts as an empty set. Requests are added to the candidate cycle until a request is repeated, triggering the creation of a new candidate cycle. If N

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

identical candidate cycles are found, proceed to Step 3. If not, iteratively test all subset combinations of requests in the group. Requests that fail all the subsets are reported as non-periodic. The message order in a candidate cycle is irrelevant.

3. *Test candidate cycles*: Calculate the minimum and maximum durations dur_{min} and dur_{max} of all the candidate cycles and verify if the difference is above the threshold dur_{thr}. If the test succeeds, return the set of requests in the cycle, its duration range dur_{min} and dur_{max}, and standard deviation dur_{std}. If the test fails, return to Step 2 and find another candidate cycle set.

The time complexity of the algorithm is exponential with the number of messages of the largest group identified in the first step of the algorithm because all subset combinations are tested. Suppose that n is the size of the largest group. In the worst case, where all the requests are not periodic, the algorithm has to test $\binom{n}{n} + \binom{n}{n-1} + \binom{n}{n-2} + \cdots + \binom{n}{1} = 2^n - 1$ combinations. However, all request combinations are not tested blindly. In Step 1, initial groups are formed by grouping requests with roughly the same numbers of occurrences. Furthermore, non-periodic requests may be discarded when searching for candidate cycles in Step 2. Note that it is assumed that periodic request–response pairs represent the majority of the traffic. During the experiments, it was observed that the Learner module always terminated after a relatively short time. In the worst case, it took 100 ms to learn 30 min of traffic on a computer with a 2.66 GHz Intel Core 2 Duo processor.

4. Evaluation

This section evaluates the proposed approach using datasets collected from real industrial control networks. The central questions are whether the empirical industrial control network traffic follows the periodicity assumptions and whether PeriodAnalyser can accurately learn the traffic characteristics. The amount of input data needed by the Learner module is also analyzed. Furthermore, the issue of whether or not flows identified as periodic change their behavior later is examined.

To perform the evaluation, a new Monitor module is incorporated to detect deviations in the traffic from the cycle models learned by PeriodAnalyser. Fig. 1 presents the evaluation approach. The traffic is first preprocessed by the Multiplexer and Tokenizer modules. During the training phase, a portion of the dataset is processed by the Learner module with the goal of identifying and characterizing the cycles. After the training phase has concluded, the Monitor module uses the cycle information to detect deviations in the remainder of the dataset. Obviously, the Monitor module works as an anomaly detector, with the cycle information describing the normal (expected) communications behavior. Section 5.1 discusses how PeriodAnalyser and the Monitor module can be used to construct an anomaly-based intrusion detection system for industrial control networks.

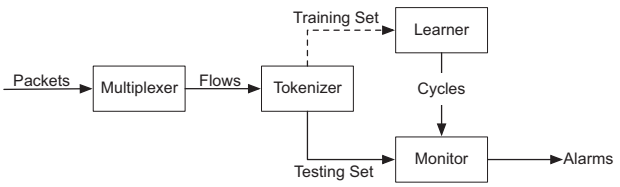


Fig. 1 – PeriodAnalyser evaluation approach.

4.1. Datasets

The experiments employed network traffic traces collected from three real-world industrial control networks, two at water treatment facilities and one at an electric-gas utility. One network used the Modbus protocol while the other two used MMS. The durations of the datasets range from one to 13 days of data and each dataset contains traffic from 11 to 45 hosts. Table 2 presents an overview of the datasets used in the experiments.

Each flow in the datasets was divided into a training set and a testing set. Unless stated otherwise, the training set consisted of the first 30 min of a flow and the testing set comprised the remainder of the traffic. Since all three datasets contained more than one day of data and the expected periodicity was in the order of a few seconds, this choice was a good compromise between an adequate number of samples and the lengths of the datasets.

4.2. Monitoring periodic behavior changes

The Monitor module observes network traffic in the testing set and raises alarms if its behavior does not match the cycles discovered by the Learner module when it processed the training set.

Fig. 2 shows the decision graph used by the Monitor module. Each cycle identified previously by the Learner module is monitored by the independent Monitor module. Hence, the first step undertaken by the Monitor module is to identify the cycle to which an incoming packet (represented by a token) belongs. If the token is an unknown request (i.e., it does not belong to any cycle), a New Request alarm is raised. If the token is a response without a previously-received matching request (identified by the pair identifier), an Unmatched Response alarm is raised.

The Monitor module can be idle or not idle. If the Monitor module is idle, then all requests belonging to the cycle have been observed in the current iteration and the currently-processed request is the potential beginning of a new iteration. The duration of the current iteration is defined as the time elapsed between the first request of the current iteration and the arrival of the currently-processed request. If the duration is greater than the allowed maximum dur_{max}, a Long Iteration alarm is raised. If the duration is less than the

Table 2 – Overview of the datasets.

Name	Hosts	Duration	Avg. Pkts/s	Avg. KB/s
Modbus	45	13 days	504.1	82.5
MMS 1	14	10 days	28.7	5.1
MMS 2	11	1.5 days	137.8	24.0

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

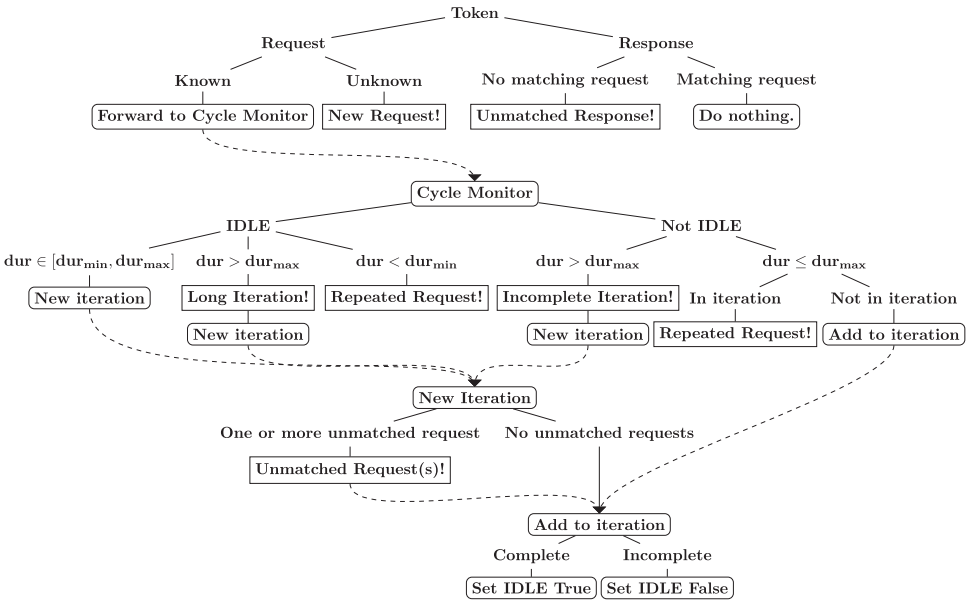


Fig. 2 – Decision graph used by the Monitor module.

allowed minimum dur_{min}, a Repeated Request alarm is raised because the request is regarded as a repetition of an already-observed request during the current iteration. Finally, if the duration is between dur_{min} and dur_{max}, a new iteration is begun. At this point, Unmatched Request alarms are raised if there are requests for which no corresponding responses are observed.

If the Monitor module is not idle, then the current cycle has not completed. The Monitor module tests if the duration of the current iteration (as defined above) exceeds the allowed maximum duration dur_{max}. If this is true, an Incomplete Iteration alarm is raised because the iteration has failed to complete in time. If not, a test is performed to check if the request has been already seen in the current iteration; if this is true, a Repeated Request alarm is raised.

Finally, the request is added to the current iteration and the state of the Monitor module changes accordingly. Note that the Monitor module is initially designated as being not idle.

As discussed in Section 3.1, small timing variations may occur. In order to make the Monitor more robust, the following dur_{min} and dur_{max} thresholds (which are more relaxed than those used by the Learner module) are employed:

$$dur_{min} = dur_{min,learned} - \max(\min_{thr}, 2dur_{std})$$
$$dur_{max} = dur_{max,learned} + \max(\max_{thr}, 2dur_{std}) \quad (1)$$

The values of the $\min_{thr}$ and $\max_{thr}$ thresholds are discussed in Section 4.4.

4.3. *Validating the communications model*

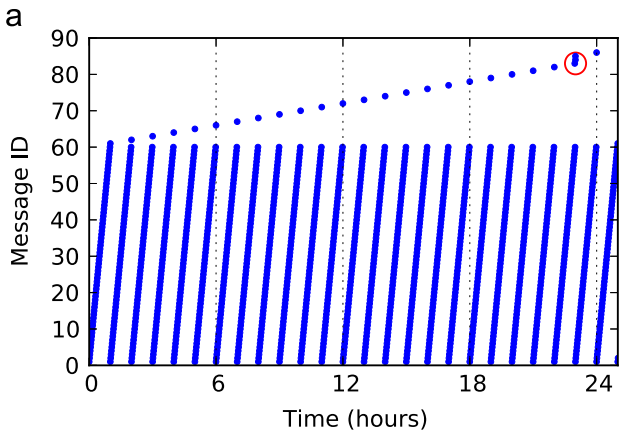
The first test seeks to verify the basic assumption that industrial control network traffic consists of a series of periodic requests and that responses can be represented by the cycle model. The next task is to distinguish between three types of flows: (i) flows that do not show any periodic behavior in the training set; (ii) flows that are periodic in the training set but present new requests in the testing set; and (iii) flows that are periodic in the training and testing sets. To perform this task, it is necessary to test whether the Learner module can find two identical candidate cycles ($N=2$). Additionally, New Request alerts raised by the Monitor module are checked. Note that this test does not consider timing-related issues reported by the Monitor module (e.g., Long Iteration alerts).

Table 3 – Number of flows per type.

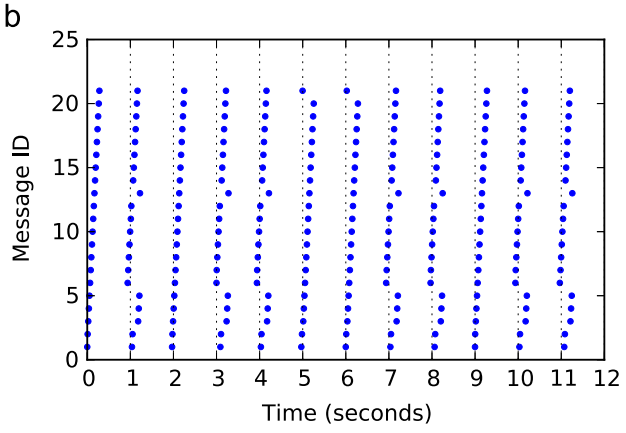
Dataset	Periodic flows	Periodic flows with new requests	Non-periodic flows
Modbus	19	4	1
MMS 1	9	0	14
MMS 2	15	0	5

Table 3 shows the results of the test. Most flows observed in the datasets fit the assumption that the traffic consists of a series of periodic requests and responses. In the Modbus dataset, only one flow is reported as non-periodic. In the combined MMS datasets, 24 of 43 flows are periodic. The high number of non-periodic flows is expected because MMS implements more advanced control schemes than Modbus. The difference also reflects the scenarios for which the two protocols were designed to be used. Modbus is a typical supervisory control and data acquisition (SCADA) protocol that supports only simple supervisory commands. In contrast, MMS is a typical distributed control system (DCS) protocol, which supports advanced commands that enable tight integration with closed process control loops. Nonetheless, more than one-half of the MMS flows consist only of series of periodic requests, demonstrating that the proposed algorithm can be used to monitor a considerable number of flows in distributed control system environments.

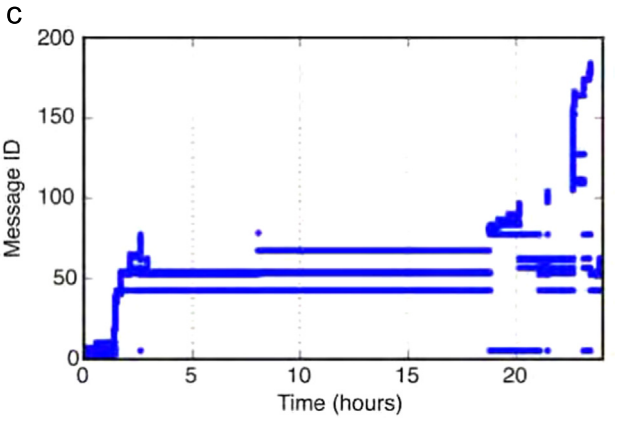
Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004



Clock synchronization flow.



Flow with unordered requests.



Flow reading unconstrained addresses.

Fig. 3 – Extracts of three flows observed in the datasets. (For interpretation of the references to color in the text, the reader is referred to the web version of this paper.)

Four of the periodic flows in the Modbus dataset contain new requests. Two flows are likely due to manual activity and the other two are clock synchronization flows. The flows with suspected manual activity are mostly periodic, except for short intervals when up to three new (read) requests are observed. Note that the false alarms generated by human activity could be avoided by correlating information from other sources, such as logs from industrial control system servers (if available) and resource management systems.

Fig. 3(a) shows an example clock synchronization flow. The clock synchronization flows contain write messages for values from 0 to 59 issued hourly (representing each minute) and from 0 to 23 issued daily (representing each hour). Such behavior obviously cannot be learned using the original 30 min training set. It requires at least 25 h of training data with $N=1$ (although the pattern has a 24 h period, the Learner module requires an additional hour to detect a repeated request, which delimits the end of a candidate cycle). However, this would not eliminate all the New Request alarms – in addition to the hourly and daily write messages, a weekly pattern of requests writing values from 0 to 7 is detected (the first one is marked with a red circle in Fig. 3(a)). Other requests repeat after nine days (not shown), suggesting even longer cycles, but the dataset is not sufficiently large to confirm this behavior.

Three Modbus flows show periodic patterns where the order in which the requests are sent is not fixed. An example is shown in Fig. 3(b). This lack of order could have been caused by a multi-threaded application (as anticipated in Section 3.1). Such periodic patterns cannot be modeled correctly by the approach of Goldenberg and Wool [13] because it assumes that the order in which requests are issued is fixed.

Fig. 3 (c) shows a typical non-periodic flow in the MMS datasets. A remarkable aspect of these flows is the presence of long periods of "well-behaved periodicity" (e.g., between 10 h and 15 h). Applying the Learner module (and later the Monitor module) to these periods reveals perfectly periodic patterns. Note that the learned requests are confirmed requests that issue read commands to an unconstrained address. The new requests are also read commands, but with slightly modified addresses. This work assumes that some of the new requests simply replace the old address without breaking the periodic pattern. Unfortunately, the semantics of the addresses are not defined by the standard (i.e., they are vendor-dependent). The verification of this assumption is a topic for future research.

Table 4 provides an overview of the different types of periodic behavior observed in the datasets by further classifying each periodic flow according to three characteristics. First, flows are classified as containing single or multiple cycles. Second, flows are classified as containing single or multiple requests per cycle; for a flow to have multiple requests, it is sufficient that at least one of its cycles contains more than one request. Third, flows are classified as single or multiple connections per flow; single connection flows represent long-lived connections with periodic requests while

Table 4 – Number of flows per type.

Cycles	Requests	Connections	Modbus	MMS 1	MMS 2
Single	Single	Single	14	2	4
Single	Multiple	Single	3	2	0
Single	Multiple	Multiple	0	5	2
Multiple	Multiple	Single	2	0	9

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

multiple connection flows represent periodic short-lived connections.

The majority of periodic flows consist of a single cycle with a single periodic request over a single long-lived connection. However, all the other combinations are also present in the datasets. This diversity in periodic behavior clearly demonstrates the need to have the two requirements of multiple cycles and periodicity at different aggregation levels discussed in Section 3.1.

4.4. *Impact of N*

This section examines the impact of the number of required candidate cycles N on the learned traffic models. It is anticipated that using more candidate cycles when patterns are being learned provides better estimates of the dur_{min} and dur_{max} thresholds, and thus, generates fewer false alarms due to normal variations in the durations of cycles.

Fig. 4 shows plots of the number of alarms a (x-axis) generated by the Monitor module versus the number of flows that generate a or less alarms (y-axis) for a given value of N. For ease of visualization, only the results for $N = 4, 8, \ldots, 24$ are shown. Note that only time-based alarms are considered. In other words, a is the sum of the Long Iteration, Incomplete Iteration and Repeated Request alarms. Other alarm types are not relevant with respect to N.

In the case of the Modbus dataset (Fig. 4(a)), the number of alarms is independent of N; in other words, all the characteristics of the cycles can be reliably learned when $N=2$. Only three flows present alarms (two of them also present the New Request alarms discussed in the previous section). The time-based alarms are caused by an "extra" iteration that does not fit the periodic behavior. Repeated Request alarms are generated for every request sent during this extra iteration. In addition, these requests cause the Monitor module to lose its synchronization with the periodic pattern, which in turn, causes the Monitor module to generate a few (false) alarms until it can synchronize with the periodic pattern. It appears that this extra iteration is triggered by manual access. In a real-world deployment that uses PeriodAnalyser to detect anomalies in industrial control network traffic (discussed in Section 5.1), these alarms could be aggregated into fewer alarms.

As seen in Fig. 4(b), the number of candidate cycles N has a large impact on the number of alarms for the MMS 1 dataset. For $N=4$ and $N=8$ (the lines overlap), five flows present more than 10^4 alarms. These alarms appear to be caused by high network delays, therefore it is necessary to use more than eight candidate cycles to obtain good values of the cycle duration thresholds. Most of the delay variability is captured when $N=12$, but the number of alarms reduces until $N=20$. At this point, most flows present ten or fewer alarms. Two of the flows still present more than 20 alarms with $N=20$. No improvements are observed when $N>20$ (the lines for $N=20$ and $N=24$ overlap).

Fig. 4 (c) shows the results for the MMS 2 dataset. Increasing the value of N reduces the number of alarms until $N=20$; and no improvements are observed when $N>20$ (again, the lines for $N=20$ and $N=24$ overlap). Even with $N=20$, four flows present a very large number of alarms – up to 10^4.

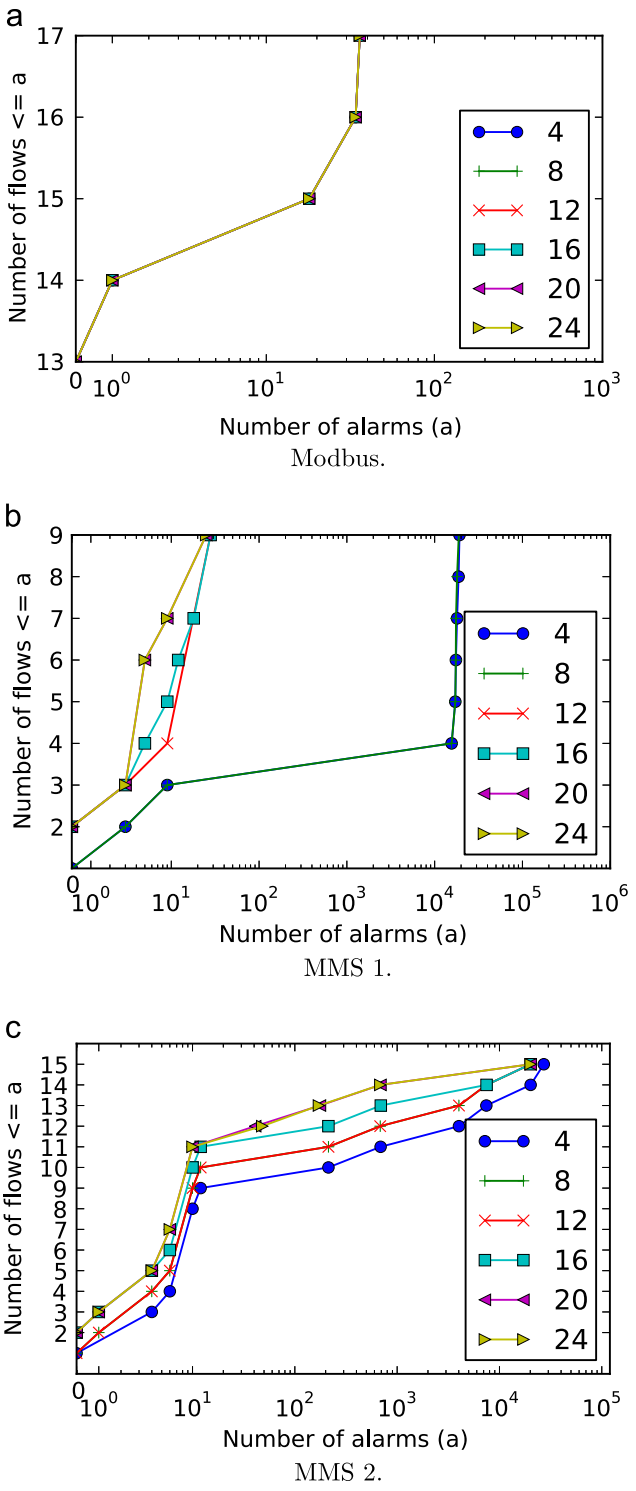


Fig. 4 – Effects of N for the three datasets.

Table 5 – Numbers of alarms in different scenarios.

Scenario	Modbus	MMS 1	MMS 2
$N=4$	89	108,599	59,704
$N=20$	89	76	21,126
Manual	–	4	309

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

Table 5 shows the numbers of alarms in three scenarios: (i) $N=4$; (ii) $N=20$; and (iii) manual adjustments of min_{thr} and max_{thr} to reduce the number of alarms. As discussed above, the alarms observed in the Modbus dataset are caused by real anomalies (i.e., deviations from periodic behavior) and cannot be eliminated by adjusting the parameters. However, in the case of the MMS 1 dataset, it is possible to reduce the number of alarms to four by manually adjusting the min_{thr} and max_{thr} thresholds to 1 s. The choice seems reasonable because the periods of the learned cycles in these flows are around 21 s.

In the case of the MMS 2 dataset, the alarms are caused by very large variations in the iteration durations; the average duration is approximately 1 s, but durations of 0.5 s are also common in these flows. Upon setting min_{thr} to 0.7 s, 309 alarms are still observed. Note that these flows together contain hundreds of TCP retransmissions and the largest numbers of Unmatched Request and Unmatched Response alarms in the datasets. This suggests that the large numbers of alarms are caused by bad link quality.

5. Practical applications

This section discusses the practical applications of PeriodAnalyser. In particular, it shows how PeriodAnalyser can be used for intrusion detection as well as for testing intrusion detection systems and evaluating the performance of industrial control devices.

5.1. *Intrusion detection*

As discussed in Section 4, PeriodAnalyser and the Monitor module can be used for anomaly-based intrusion detection. In this case, the Learner module constructs a cycle-oriented normal model of industrial control system communications from a training dataset. The Monitor module then detects deviations from the learned behavior in live network traffic. Alarms raised by the Monitor module indicate anomalies. However, it is important to note that not all variations in periodic behavior are relevant from a security perspective. For example, the time-based alarms may only indicate performance issues, not security threats. In fact, most of the alarms observed for the MMS 2 dataset were due to small variations in the durations of iterations, which were likely caused by normal network delays. Optionally, time-based alarms could be disabled in such instances.

The three datasets used in this research were free of attacks. At this time, no empirical datasets containing malicious industrial control network traffic are available. Hence, the following discussion is based on some of the general attack classes proposed in [17]. Where available, concrete types of attacks are presented from an open access list of real-world attack signatures used in the Quickdraw Intrusion Detection System [10]. The list includes signatures for protecting Modbus TCP and DNP3 [9], two common SCADA communications protocols.

- *Information gathering attacks*: These attacks, which typically precede other attacks, are used to gather as much knowledge as possible about the targeted system. A typical way to acquire this information is through scans such as the Modbus TCP Function Code Scan. In general, attacks of this type require a large number of parameters (addresses, ports, function codes, etc.) to be tested. Therefore, they likely make use of requests that are not observed during the learning phase and, therefore, trigger New Request alarms. Even if it is possible to craft an attack using the learned periodic requests, the scan would still be identified by Repeated Request alarms.
- *Denial-of-service attacks*: These attacks prevent legitimate users from accessing services or negatively impact the provision of services. For example, the DNP3 Unsolicited Response Storm attack attempts to overload a DNP3 device by sending a number of unsolicited response packets, which are normally used to report alarms. Once again, if the attack uses new requests, they would be identified immediately. The attacker could attempt to repeat the messages from a cycle rapidly, but this would also generate a large number of Repeated Request alarms. While a small number of repeated requests do not pose a security threat, a large number may indicate an attempt to overload the receiver by rapidly repeating requests in a cycle. Note that such attacks are not detected by the approach proposed by Goldenberg and Wool [13].
- *Network attacks*: These attacks manipulate network protocols. For instance, the Modbus TCP Clear Counters and Diagnostic Registers attack uses a single packet with a specific function code to clear counters and diagnostic registers in a device in an attempt to avoid detection. The Modbus TCP Slave Device Busy Exception Code Delay attack answers every request with a "device is busy" message, preventing a response timeout. Such attacks rely on the use of uncommon requests, which would readily be identified by New Request alarms. In general, New Request alarms also provide powerful protection against data injection attacks.
- *Buffer overflow attacks*: These attacks attempt to gain control of a process or crash it by overflowing its buffer. For example, the Modbus TCP – Illegal Packet Size (Possible DoS) attack sends a single packet with an illegal packet size, exploiting a bug in the implementation of the protocol stack. The attacks rely on forging invalid packets that are, by definition, not commonly used. Thus, they also generate New Request alarms.
- *Anomalies not related to malicious activity*: Alarms created by the Monitor module can also be used to detect anomalies that are not or only indirectly related to malicious activity. For example, a Long Iteration alarm indicates that a sender has become inactive. However, SCADA applications are designed to deal with delays [2], so accepting delays that are longer than those learned from the data might be acceptable or even desirable. Finally, while a small number of Incomplete Iteration and Unmatched Request/Response alarms do not pose a serious threat (they are likely due to a network performance problem or a probe causing packet loss), a large number of alarms indicate that the receiver has become inactive. Note, however, that the threshold at which these alarms become a security issue is dependent on the environment. For instance, water processes are

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

considerably slower than electric processes, so a water treatment system might tolerate larger delays.

In summary, the proposed approach provides strong protection against data injection. Only requests commonly used during normal behavior can be used in attacks, otherwise New Request alarms would be issued. Furthermore, attacks created using normal requests must be sent in a manner that does not differ considerably from normal periodic behavior, otherwise they will cause Repeated Request alarms.

5.2. *Synthetic trace generation*

The previous section discussed how the models learned by PeriodAnalyser could be used to identify anomalies in observed sequences of messages for the purpose of intrusion detection. In contrast, this section discusses how the models can also be used to output sequences of messages. The goal is to generate synthetic message traces that are realistic (i.e., those that have properties close to what are observed in real systems).

Like other models, the models obtained by PeriodAnalyser are abstractions of reality and, hence, represent the behavior of a real system only to a certain degree. The Tokenizer module abstracts protocol-dependent information by extracting specific fields from requests and responses to generate protocol-independent tokens (see Table 1). While enough information is maintained to uniquely identify requests and to match requests to responses, the contents of the responses are mostly ignored. To use a real-world example, the model can be used to capture how often the water level of a tank is checked, but not the actual water level in the tank.

In the PeriodAnalyser models, a learned cycle carries information about the sources and destinations of the exchanged messages, the numbers and types of requests that are allowed in the cycle, and the minimum duration, maximum duration and standard deviation. Consequently, a learned model enables the generation of synthetic traces that reflect the randomness of the processes observed in the real system. The order of the requests and their timing can be permuted according to the learned cycle information.

Such synthetic traces are useful for various purposes. In the context of cyber security and intrusion detection, a major challenge for researchers is the lack of publicly-available traces of industrial control network traffic, primarily because asset owners are reluctant to provide real data for reasons of sensitivity. Synthetic traces generated using the PeriodAnalyser models can be combined with attack traffic to obtain the ground truth for testing and evaluating intrusion detection systems. The fact that the contents of response messages are not modeled is not necessarily a drawback. Many intrusion detection systems, for example, flow-based intrusion detection systems [23], as well as, to some extent, the intrusion detection system discussed in Section 5.1, do not rely on deep packet inspection.

A second application area is the creation of load generators for evaluating the performance of control system components in simulators. An empirical data trace is often inadequate for this task because it reflects the behavior of one specific system. In contrast, the proposed model-generated synthetic traces are scalable and flexible. For example, it is possible to "clone" the cycle model of a programmable logic controller with different source and destination addresses, thereby simulating a large industrial control network. Furthermore, the distributions of durations could be varied, for example, to simulate the behavior of an industrial control system that uses a high-latency communications network.

6. Conclusions

The PeriodAnalyser tool presented in this paper embodies an automated approach for learning traffic models from industrial control network traffic. The novel approach addresses the limitations of existing approaches by directly exploiting the periodicity inherent in industrial control network traffic. The "semantic gap" of spectral analysis is avoided by identifying the messages that produce periodic behavior. Also, timing information is captured directly, making it possible to identify and monitor the frequencies at which messages are exchanged. This enables the approach to automatically learn multiple periods and, therefore, adapt to a variety of industrial control system environments.

The approach was evaluated using three real-world traffic traces that covered the Modbus and MMS protocols. The results show that most flows of SCADA protocols such as Modbus can be accurately modeled by the approach. Although more flow variations are encountered in distributed control protocols such as MMS, a large number of flows comprise periodic requests that are also accurately modeled by the approach.

The traffic modeling approach has two important applications. The first application is intrusion detection, which provides two layers of protection. One layer of protection is a whitelist created by learning the polling requests issued to field devices, which prevents data injection attacks by third parties. The second layer, which involves learning the rate at which requests are sent, helps prevent denial-of-service attacks that attempt to overload field devices with (whitelisted) requests. In addition, the learned rate also helps detect when specific requests are not sent for a long period of time (e.g., when the process responsible for issuing the requests becomes inactive). The second key application provided by the traffic modeling approach is the generation of meaningful synthetic traffic traces. These traffic traces can be used as ground truth to test intrusion detection systems or as scalable loads to evaluate the performance of industrial control devices.

Future research will focus on enhancing the Learner and Monitor modules to ascertain if assumptions can be made about periodic request cycles (e.g., checking if requests are sent in a fixed order). Also, research will attempt to extend the approach and the implementation to handle other industrial control protocols such as DNP3 and IEC 60870-5-104.

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004

Acknowledgment

This research was partially supported by the European Union 7th Framework Program via Projects FP7-SEC-607093-PREEMPTIVE and FLAMINGO, a Network of Excellence Project (318488).

REFERENCES

[1] O. Argon, Y. Shavitt and U. Weinsberg, Inferring the periodicity in large-scale Internet measurements, *Proceedings of the Thirty-Second IEEE Conference on Computer Communications*, pp. 1672–1680, 2013.

[2] D. Bailey and E. Wright, *Practical SCADA for Industry*, Newnes, Oxford, United Kingdom, 2003.

[3] R. Barbosa, R. Sadre and A. Pras, A first look into SCADA network traffic, *Proceedings of the IEEE Network Operations and Management Symposium*, pp. 518–521, 2012.

[4] R. Barbosa, R. Sadre and A. Pras, Difficulties in modeling SCADA traffic: A comparative analysis, *Proceedings of the Thirteenth International Conference on Passive and Active Measurement*, pp. 126–135, 2012.

[5] R. Barbosa, R. Sadre and A. Pras, Towards periodicity based anomaly detection in SCADA networks, *Proceedings of the Seventeenth IEEE Conference on Emerging Technologies and Factory Automation*, 2012.

[6] A. Broido, E. Nemeth and K. Claffy, Spectroscopy of private DNS update sources, *Proceedings of the Third IEEE Workshop on Internet Applications*, pp. 19–29, 2003.

[7] M. Caselli, E. Zambon and F. Kargl, Sequence-aware intrusion detection in industrial control systems, *Proceedings of the First ACM Workshop on Cyber-Physical System Security*, pp. 13–24, 2015.

[8] M. Cheminod, L. Durante and A. Valenzano, Review of security issues in industrial networks, *IEEE Transactions on Industrial Informatics*, vol. 9(1), pp. 277–293, 2013.

[9] K. Curtis, A DNP3 Protocol Primer, DNP Users Group, Calgary, Canada, 2005.

[10] Digital Bond, Quickdraw SCADA IDS, Sunrise, Florida ⟨www.digitalbond.com/tools/quickdraw⟩, 2016.

[11] G. Ericsson, Cyber security and power system communications – Essential parts of a smart grid infrastructure, *IEEE Transactions on Power Delivery*, vol. 25(3), pp. 1501–1507, 2010.

[12] S. Floyd and V. Paxson, Difficulties in simulating the Internet, *IEEE/ACM Transactions on Networking*, vol. 9(4), pp. 392–403, 2001.

[13] N. Goldenberg and A. Wool, Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems, *International Journal of Critical Infrastructure Protection*, vol. 6(2), pp. 63–75, 2013.

[14] G. Gu, J. Zhang and W. Lee, BotSniffer: Detecting botnet command and control channels in network traffic, *Proceedings of the Fifteenth Annual Network and Distributed System Security Symposium*, 2008.

[15] H. Hadeli, R. Schierholz, M. Braendleand and C. Tuduce, Leveraging determinism in industrial control systems for advanced anomaly detection and reliable security configuration, *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, 2009.

[16] J. Han, G. Dong and Y. Yin, Efficient mining of partial periodic patterns in time series database, *Proceedings of the Fifteenth International Conference on Data Engineering*, pp. 106–115, 1999.

[17] S. Hansman and R. Hunt, A taxonomy of network and computer attacks, *Computers and Security*, vol. 24(1), pp. 31–43, 2005.

[18] International Standards Organization, ISO/IEC 9506 – Manufacturing Message Specification (MMS), Geneva, Switzerland, 1990.

[19] D. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Boston, Massachusetts, 1969.

[20] S. Laxman and P. Sastry, A survey of temporal data mining, *Sadhana*, vol. 31(2), pp. 173–198, 2006.

[21] Modbus-IDA, Modbus Messaging on TCP/IP Implementation Guide V1.0b, Hopkinton, Massachusetts ⟨www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf⟩, 2006.

[22] R. Sommer and V. Paxson, Outside the closed world: On using machine learning for network intrusion detection, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 305–316, 2010.

[23] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras and B. Stiller, An overview of IP flow based intrusion detection, *IEEE Communications Surveys and Tutorials*, vol. 12(3), pp. 343–356, 2010.

[24] J. van Splunder, Periodicity Detection in Network Traffic, M. Sc. Thesis, Institute of Mathematics, University of Leiden, Leiden, The Netherlands, 2015.

[25] J. Zhang and A. Moore, Traffic trace artifacts due to monitoring via port mirroring, *Proceedings of the Workshop on End-to-End Monitoring Techniques and Services*, 2007.

Please cite this article as: R.R.R. Barbosa, et al., Exploiting traffic periodicity in industrial control networks, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.02.004