On the Use of K-NN in Intrusion Detection for Industrial Control Systems

Pedro Silva[1], Michael Schukat[2]

[1]*Discipline of Information Technology*
College of Engineering & Informatics
National University of Ireland, Galway
Galway, Ireland
Email: gygabyte@gmail.com

[2]*Discipline of Information Technology*
College of Engineering & Informatics
National University of Ireland, Galway
Galway, Ireland
Email: michael.schukat@nuigalway.ie

***Abstract*— Cyber security in industrial control systems and critical infrastructure is becoming increasingly important. However, many installations still rely on legacy communication protocols like Modbus that provide no active protection against potential cyber attacks; among other things, these protocols do not support source authentication or payload encryption / authentication.**

Intrusion detection systems (IDS) provide a passive form of cyber security as they monitor network or system activities for malicious activities or policy violations. As passive and non-intrusive safeguards they are particularly useful for the above legacy systems.

However, IDS in ICS require advanced approaches to intrusion detection, which go beyond conventional blacklisting / whitelisting. This paper examines a new technique, which is based on using the K-Nearest Neighbour scoring algorithm to discover unsolicited data patterns in ICS network traffic.

***Keywords* – Intrusion Detection, Industrial Control System, K-Nearest Neighbour, Pattern Whitelisting**.

I INTRODUCTION

The discovery of Stuxnet in 2010 [2] resulted in a huge media attention, but it only highlighted a problem that was already known for a decade: the vulnerability of Industrial Control Systems (ICS).

ICS is a general term that encompasses several types of control systems used in industrial production. Such systems are also widely deployed in large and (mission-) critical infrastructures including power grids and water / gas distribution systems. The failure of such systems (for example because of a cyber attack) can potentially cause disastrous events.

The security of Industrial Control System has been a major concern over recent years, as failures in such systems may put at risk the health and safety of entire communities [1]. In recent years we have seen a noteworthy number of incidents with Stuxnet being the most noted one.

Industrial Control Systems deployments last for many years and many of them are consequently based on legacy protocols that were designed in the 1980s and 90s. At that time it was not expected to have these systems connected to other networks as local office networks or directly to the Internet. As a result these protocols do not provide any form of secure or authenticated communications. However, as times evolved ICS systems are becoming accessible from the outside world but no security measures are applied which makes them highly vulnerable to attacks [3].

Upgrading these systems can be challenging since they are running the same software and hardware for decades. In some cases in order to update a single component of a control system, it is also needed to update the underlying operating system which might not be possible without also upgrading the hardware as well. These dependencies make any small attempt to move the system to a more secure version a tremendous challenge. These difficulties often discourage operators from updating the system or implement security measures that require newer versions. There are also other cases where operators are running proprietary software that system specification or even the source code is no longer known. In these cases is nearly impossible to update the system.

Another problem is the issue of cyber-fragility, e.g. the non-ability of some complex ICS to compensate for small changes in their operational environment. For example, it has been reported that small changes in communication latency times as a result of software / firmware upgrades can bring down an entire control system [4].

In this context, Intrusion Detection Systems (IDS) can play an important role to make ICS more secure. A typical IDS for networks usually does not require any change in the control system. An IDS is a passive component that captures network traffic to be analysed. The result of that analysis will determine if an intrusion has been detected.

Typically the network traffic of an ICS is highly periodic which usually leads to highly predictable and deterministic network traffic patterns. The work presented in this paper leverages this periodicity to determine anomalies in network traffic using k-NN algorithm.

The paper is organized as follows: In Section II, we review related work in Intrusion Detection Systems for Control Systems. In Section III, we explain our intrusion detection approach. In Section IV, we discuss possible attack vectors that we believe our approach will be able to detect. In Section V, we present our conclusions. In Section VI, we discuss future work.

II Related Work

Cheung et al. [5] proposed a model-based approach to detect intrusions in SCADA networks. These models specify the expected behaviour of the system and the IDS generates an alarm when deviations are found. Several models approaches are discussed. Protocol-level models are introduced to find deviations in specific fields of Modbus protocol such as function codes. Dependent fields models are also discussed, i.e., a value of a field may depend of the value of another field. These models have been implemented using Snort which is an open-source network detection system. It is also discussed more complex models that possible involves multiple Modbus requests and responses. In this case, the authors preferred to write a custom IDS due to Snort limitations and complex architecture. However, it is stated that this IDS will be based on formal models. Prior knowledge about the deployed ICS infrastructure will be required to these models approaches be effective which in some cases is not accurate.

Barbosa et al. [6] presented an approach based on the traffic periodicity of SCADA networks. They realized there are some SCADA networks that its normal behaviour consists only of periodic bursts of packets generated by polling mechanisms. It is assumed that a considerable number of intrusions attempts or anomalies may disrupt the traffic periodicity. There are attacks such as Denial of Service that can be easily detected as the attacker will send a large volume of packets in a short period of time. Attacks that do not generate large volume and only need a few packets might not be detected. However, the attack can be detected by its effect in the SCADA system. A Buffer overflow attack optimally only requires one malicious packet, but often the targeted process crashes. This event will cause interference in the normal traffic patterns which will be detected. The proposed approach consists of four modules. The first module captures network traffic in a passive mode. This module filters out all packets that are not relevant to SCADA processes. The second module will construct the network flows using the server-side transport port to isolate traffic. Additional attributes might be needed if the service is receiving requests from more than one client. The flows are created by sampling a fixed interval P. The sample frequency SF is defined as $SF = 1 / P$. For each sample the number of captured packets is stored. Before initiate detection, the periodicity learning module will extract the frequency of the periodic bursts and their size. This step is performed offline and results validated by operators. As detected flows are matched to known periodic burst it will be analysed in order to find anomalies. An alarm is raised when an anomaly is detected. These alarms are also processed by the periodicity learning module to support adaptive learning. It is not stated how this adaptive learning is implemented. A Proof of Concept is explained to demonstrate the feasibility of the approach. A spectrogram-based anomaly detection is implemented and data capture is emulated by using previously collected data. The results show that the use of a spectrogram is feasible but as the analysis is performed manually, the entire system is not viable without being automated. Also, it is concluded the algorithm is too sensitive to noise such as networks delays.

III Our Approach

We assume that ICS network traffic is highly periodic. This observation is based on the nature of such systems where usually Programmable Logic Controllers (PLC) are used to control automated tasks which, in most cases, are executed periodically.

a) K-Nearest Neighbour

Case-Based Reasoning (CBR) is the process to solve novel problems (case) with solutions of similar solved problems (case base) [7]. This process is accomplished by performing the following functions: *Retrieve*, *Reuse*, *Revise* and *Retain*. K-Nearest Neighbour is a scoring algorithm commonly used in CBR for retrieving similar cases.

The K-Nearest Neighbour (k-NN), in a CBR context, allows to query a case base in order do find cases which are the most similar to the issued query [7]. In our context, each case is a packet captured from the network. The similarity of each packet in the case base to the queried packet is calculated through a *similarity function* [7] and is represented as a real number in $[0, 1]$. The retrieved cases can be the k most similar cases or a threshold can be defined and the cases which surpass this threshold are considered similar.

A case is often represented as a set of attribute-value pairs [7]. To each attribute is defined a *local* similarity function $lsim(a_i,b_i)$ where a_i is an attribute of a case. A weight can be defined for each attribute and it is defined by the function $w(a_i)$ for a given attribute a_i. The weight allows changing the importance of an attribute. A global similarity is calculated as a weighted average of the local similarities. Equation (1) show the function used to calculate the similarity of two packets p_1 and p_2.

$$sim(p_1, p_2) = \frac{\sum_{i=1}^{N} lsim(a_i(p_1), a_i(p_2)) * w(a_i)}{\sum_{i=1}^{N} w(a_i)} \quad (1)$$

b) Intrusion Detection Algorithm

Our approach is based on ICS network traffic being highly periodic. Due to this fact, these networks have a tendency to be predictable and deterministic [6]. This also means that it expected to have very similar packets crossing the network. We believe that most of the times these similar packets will be sent based in a pattern which might be possible to identify by using the k-nearest neighbour (k-NN) method. The k-NN method searches by performing queries to a given set of objects and returns a set with a score for each object. This paper proposes the application of this method to find similar packets crossing a network and with that identify patterns. As mentioned earlier each packet needs to be represented as a set of attributes. A similarity function and weight is also associated (see Figure 1).

Attribute	Sim. function	Weight
Source MAC Address	Equal	1.0
Dest. MAC Address	Equal	1.0
Source IP Address	Equal	1.0
Dest. IP Address	Equal	1.0
Source TCP Port	Equal / ModbusTCPPortEqual	1.0
Dest. TCP Port	Equal / ModbusTCPPortEqual	1.0
Modbus ProtoId	Equal	1.0
Modbus Length	Equal	1.0
Modbus Unit ID	Equal	1.0
Modubus Function Code	Equal	1.0

Figure 1: Attributes of an packet and correspondent similarity function and weight

The *Equal* function (see Figure 2) is used on all attributes. However, when calculating the similarity of two given packets, different similarity functions of TCP Port attributes are used, depending on the context.

```
Equal(att1, att2) {
  if(att1 == att2)
    return 1.0
  else
    return 0.0
}


ModbusTCPPortEqual(att1, att2) {
  if(att1 == 502 || att2 == 502)
    return Equal(att1, att2)
  else
    return 1.0
}
```

Figure 2: Local similarity functions

In a TCP connection, when a client connects to a server, a random port number is associated to the client. This port is the Source TCP Port. If this connection, for some reason, terminates and a new one is established, the Source TCP Port most likely will be different. Therefore, when finding similar packets in the whitelist, the *ModbusTCPPortEqual (*see Figure 2) function is used to address this fact or otherwise actual similar packets may not be considered similar. When searching for similar packets in other lists such as list of live captured packets, *Equal* function is used.

The approach proposed by this paper works as follows. First, the system needs to capture normal traffic which will be the whitelist. There can be more than one whitelist. From the whitelist we also build through manual observation a set of models which allows the algorithm to perform a model-based detection when needed. These models do not need a formal specification of the ICS which we believe is an advantage. The live capture is continuous but it is not processed as a stream. A time window *t ms* needs to be set and the system delivers to the intrusion detection algorithm a list *l* with packets per iteration *t*. The number of packets may differ between iterations since these blocks are delimited by time. The intrusion detection algorithm performs the following operations per each packet p_i in *t*. First, tries to find if a packet similar to p_i has been processed. It does this by running a k-NN query to a list of packets previously processed. If it can find similar packets, then it does not process p_i and jumps to p_{i+1}. If no similar packet is found, then it adds p_i to that list. Next, the algorithm will perform a k-NN query to list *l* and return a list sim_l with all similar packets to p_i. Next step is performing the same k-NN query to the whitelist and get a list sim_{wl} with all similar packets to p_i. If sim_{wl} is empty, then we can infer that the p_i is not known and will be considered as an anomaly. The algorithm signals the anomaly and jumps to p_{i+1}.

```
cmp_seqs(live_diffs[], wl_diffs[]) {
  for(i = 0; i + LEN(live_diffs) ≤
LEN(wl_diffs); i++) {
    for(j = 0; j < LEN(live_diffs); j++) {
      if( |live_diffs[j] - wl_diffs[i+j]| >
MARGIN )
        // time spans do not match
        break
    }
    if(j == LEN(live_diffs) – 1) {
      // all live time spans match
      return NORMAL_TRAFFIC
    }
  }
  // could not match time spans
  return INTRUSION_DETECTED
}
```

Figure 3: Pseudo-code for detecting anomalies/intrusions

If sim_{wl} is not empty, the algorithm now has two lists of similar packets: sim_{l} and sim_{wl}. Each packet p_j in these lists has a timestamp t generated when p_j was captured. Next step is calculating using function $CalcSeq(sim_x)$ the lists $diff_{l}$ and $diff_{wl}$, from, respectively, sim_{l} and sim_{wl}, which will contain the timestamp difference between two consecutive packets p_j and p_{j-1}. Therefore, given funtion $t(p_j)$ that returns the timestamp of a packet p_j we have the function $CalcSeq(sim_x) = \{ t(p_1) – t(p_0), \ldots t(p_j) – t(p_{j-1}) \}$, for each p in sim_x. Hence, $diff_{l} = CalcSeq(sim_{l})$ and $diff_{wl} = CalcSeq(sim_{wl})$. Next step is trying to find a subset in $diff_{wl}$ that matches $diff_{l}$. Basically, the algorithm (see Figure 3) is trying to determine if the pattern observed in $diff_{l}$ can be found in the $diff_{wl}$.

The timestamps do not have to be exactly the same. We allow the definition of a margin in *ms* which tries to mitigate some slight differences in the pattern caused by other factors such as network delays and clock imprecisions.

If no match can be found, then the algorithm will try to find a match in the models defined. A model is defined by a list of possible time ranges for a particular packet. Each item in the list has a high range and a low range values. A set of these models for a particular packet defines a connection model. A packet can be associated to more than one connection model. We refer to a set of connection models as an ICS model. The algorithm looks for a sequence of models that matches the time difference list of the live capture.

Software clock imprecisions of modern Operating Systems, such as Linux, may impact the ordering of packets crossing the network over time [8]. This is based on the assumption that packets might be sent within a multithreaded environment where multiple threads share connections and send different packets among them. Usually *systems calls* such as *select(2)* are used by developers to control network operations which creates the frequency patterns. Our algorithm using k-NN determines this frequency pattern for each packet and, therefore, circumvents the possible out of order of packets issue caused by clock imprecisions. However, if different *threads* send similar packets, then these frequency patterns can also change over time and k-NN will not be able to solve this issue.

IV ATTACKS

In this section a number of attacks are discussed and also explained how our approach could detect them.

Function name	Function code
Read Discrete Inputs	2
Read Coils	1
Write Singe Coil	5
Write Multiple Coils	15
Read Input Register	4

Figure 4: Modbus function codes (incomplete)

An attacker can create or use tools that leverages knowledge about PLC devices. *Modbus* [9] and *Modbus/TCP* [10] are open protocols that device vendors implement and publish their configurations. Figure 5 illustrates the specification of a *Modbus/TCP* frame and Figure 4

Name	Length	Function
Transaction Identifier	2	Identification of a Modbus request/response transaction
Protocol Identifier	2	Zero for Modbus/TCP
Length	2	Number of following bytes
Unit identifier	1	Identification of a remote slave
Function code	1	Function code as in other variants
Data Payload	N	Data of response or request

Figure 5: Modbus TCP protocol specification

To demonstrate the attacks we assume the following network topology illustrated in Figure 6. The network consists of a *Modbus Master* which is a SCADA Server, a *Modbus Slave* which is a PLC, our IDS running and this network is connected to other Local Area Network (LAN).

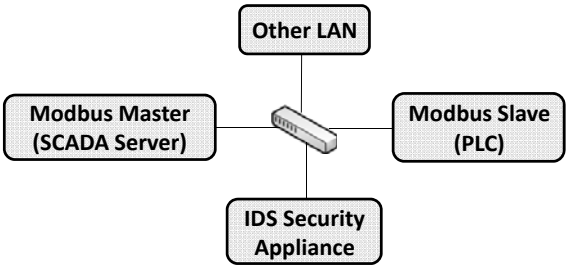

Figure 6: Network topology or our scenario

The SCADA Server issues requests to the PLC to read registers from it periodically (see Figure 7). The IDS system is passively capturing all Modbus packets. Hosts in other LAN have direct access to the SCADA network. We will now discuss four attack vectors and explain how our approach will be able to detect and flag them as anomalies or intrusions.

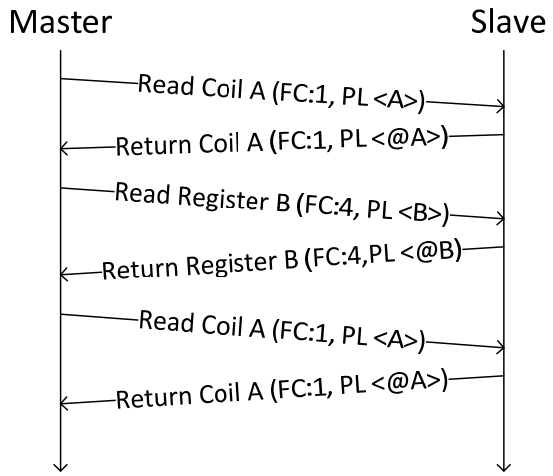

Figure 7: Simple Modbus Master/Slave interaction

a) PLC Scan

This attack is based on an attacker sending read requests in order to identify PLC devices in the network (see Figure 8). In our scenario, the SCADA system is connected to a LAN with unknown devices and topology. If an attacker sends these reconnaissance packets from an unknown device, our IDS would trigger an alert since these packets from an unknown IP Address are not in the whitelist. But, even if the attacker could somehow gained access to our SCADA Server and send these packets from a known IP Address, it would most likely generate packets with an unknown pattern which would trigger an alert as well.

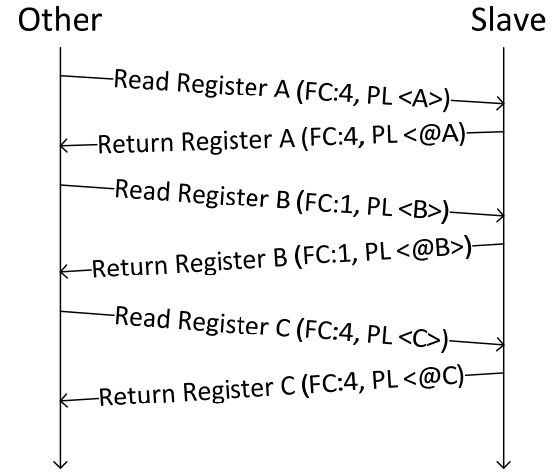

Figure 8: PLC Scan attack by an intruder

b) Force Listen Mode Only

This attack is based on sending the *Force Listen Mode* function to an ICS Slave device which will become silent and ignore any incoming request (see Figure 9).

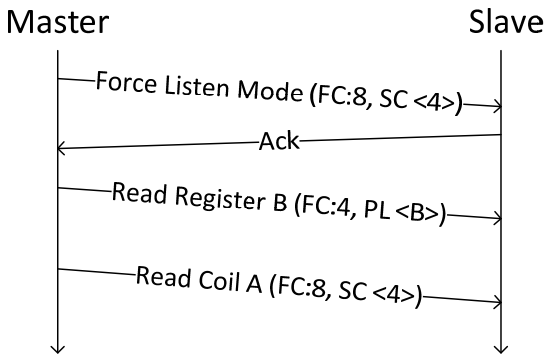

Figure 9: Force Listen Mode attack

We would be able to detect this attack because this specific Modbus function code will never be in the whitelist since it is a dangerous request and should never be sent within normal operation.

c) Process Manipulation

This attack is based on an attacker that gained access to our SCADA server and could change the behaviour of running processes.

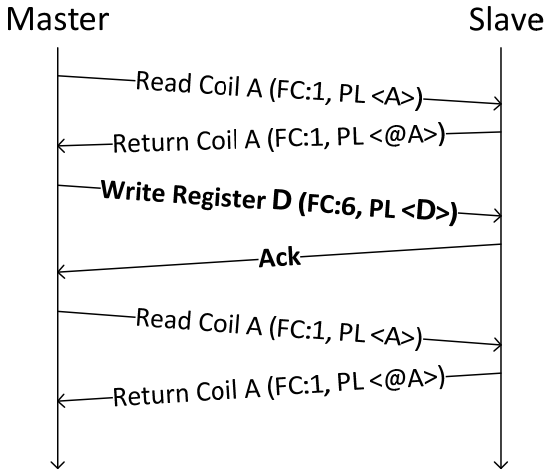

Figure 10: Process manipulation where the attacker replaces a read request by a write register request

This can also be done, without gaining privileged access of the operating system, if a process exposes an interface such as a Web Interface which might allow the manipulation of Modbus requests. In this scenario, the malicious packets will be sent from a known IP Address of the system, but this manipulation will disrupt the traffic pattern which can be detected (see Figure 10). However, we should note that if this manipulation occurs on the *Data Payload* of Modbus protocol, our IDS will not detect it, because we are not still considering the *payload* in the similarity function.

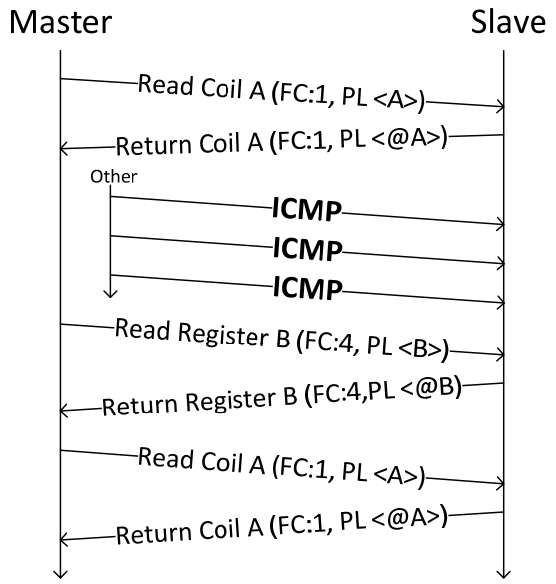

Figure 11: ICMP Flood attack as a mean to achieve DoS

d) Denial of Service (DoS)

This attack is based on sending a large number of packets to a server to prevent legitimate clients accessing the provided service. In our scenario, this could be achieved by, for example, sending a large number of ICMP packets *(ICMP Flood)* to a PLC and prevent the SCADA Server of accessing it. This would cause a disruption in the communication and thus causing a disturbance in the network traffic pattern (see Figure 11). Our IDS would be able to detect such disturbance.

V CONCLUSION

This paper presented a novel approach to perform intrusion detection in a context of ICS. As far as we are aware of there is no other work where an inspection is performed packet by packet, taken into account patterns including timestamp differences between packets. We also discussed possible attacks that our approach will be able to detect. Preliminary results demonstrated the feasibility of the proposed approach. However, we should note that it has only been tested against only one dataset.

VI FUTURE WORK

In future work, support to multiple connections between two hosts will be added. Also, when finding a match between models, the IDS allow a sequence of any arbitrary order which might flag a *true positive* as a *false negative*. The k-NN query runs by processing sequentially all packets in the case base which has a complexity of *O(n),* where *n* is the number of cases [7]. This might be impeditive if *n* is a huge number. We plan to study the feasibility of adopting other indexing strategies such as *k-d tree* (binary tree).

REFERENCES

[1] Macaulay, T, Singer, B.L. (2012), Cybersecurity for Industrial Control Systems, *Auerbach Publications*, ISBN 1439801967

[2] http://www.symantec.com/connect/blogs/stuxnet-using-three-additional-zero-day-vulnerabilities (Accessed on 2014, March)

[3] Knapp, E. (2011), Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems, *Syngress*, ISBN 1597496456

[4] Langner, R. (2011). Robust Control System Networks: How to Achieve Reliable Control After Stuxnet. *Momentum Press*, ISBN 1606503006.

[5] Cheung, S., Dutertre, B., Fong, M., Lindqvist, U., Skinner, K., & Valdes, A. (2007, January). Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA security scientific symposium* (pp. 1-12).

[6] Barbosa, R. R. R., Sadre, R., & Pras, A. (2012, September). Towards periodicity based anomaly detection in SCADA networks. In *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on (pp. 1-4)*. IEEE.

[7] Lopez De Mantaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., ... & Watson, I. (2005). Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 20(03), 215-240.

[8] Etsion, Y., Tsafrir, D., & Feitelson, D. G. (2003, June). Effects of clock resolution on the scheduling of interactive and soft real-time processes. In *ACM SIGMETRICS Performance Evaluation Review* (Vol. 31, No. 1, pp. 172-183). ACM.

[9] http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (Accessed on 2014, March)

[10] http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (Accessed on 2014, March)