Communication Pattern Anomaly Detection in Process Control Systems

Alfonso Valdes
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
Email: alfonso.valdes@sri.com

Steven Cheung
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
Email: steven.cheung@sri.com

***Abstract*—Digital control systems are increasingly being deployed in critical infrastructure such as electric power generation and distribution. To protect these process control systems, we present a learning-based approach for detecting anomalous network traffic patterns. These anomalous patterns may correspond to attack activities such as malware propagation or denial of service. Misuse detection, the mainstream intrusion detection approach used today, typically uses attack signatures to detect known, specific attacks, but may not be effective against new or variations of known attacks. Our approach, which does not rely on attack-specific knowledge, may provide a complementary detection capability for protecting digital control systems.**

I. INTRODUCTION

Intrusion Detection Systems (IDS) using anomaly detection (AD) techniques are not widely deployed in enterprise systems, because such systems typically exhibit highly variable behavior. As such, AD systems, particularly those based on learning normal system activity and alerting on abnormal activity, often alert on activity that is unusual, but not malicious, while failing to alert on malicious activity that recurs frequently enough to not appear unusual.

By contrast to enterprise systems, process control systems often exhibit regular and predictable communication patterns, which can be leveraged in an AD system. An attack launched against a process control network may exhibit communication patterns quite different from those observed during normal operations. In our earlier work, we have demonstrated that these regularities can form the basis of a model-based IDS in control systems, where much of the expected behavior of the system can be coded into a fairly compact ruleset/model, which complements misuse detection rules used for detecting known malicious activity. Because developing the models to specify the expected system behavior by hand is error-prone and time-consuming, this paper presents a learning-based communication pattern anomaly detection approach for process control systems.

Our approach involves learning network communication patterns in process control networks by passively monitoring network traffic. Specifically, our IDS employs network flow information such as connection endpoints (i.e., source and destination IP addresses and port numbers), the rate of packet flow between network endpoints, and the set of hosts with which a host communicates. The IDS maintains a database of recent and historical network flow profiles observed in process control networks. A flow record is generated or updated as packets are observed. Detected network flow patterns are then evaluated against the learned historical norms. An observed pattern can either match an existing historical flow profile through reinforcement learning, or start a new pattern exemplar. The pattern exemplars are effectively different modes of observed activity, so our system does not require attack-free training data. By default, the system alerts on observed flow patterns that are statistical exceptions to the learned norms. We are interested in anomalies such as new network flows (with estimates of how unlikely it is to observe a new flow), significant changes in flow rates, and the absence of expected network flows. These anomalies may correspond to network probing attacks, propagation of malware, introduction of rogue master or slave devices, flooding-based denial-of-service attacks, or attacks that cause host or service failure.

The principal contribution of this paper is a demonstration that anomaly detection, and specifically methods based on adaptive learning, can provide a useful intrusion detection capability in process control networks. This is in contrast to the efficacy of these methods in enterprise settings, where highly variable behavior leads to inadequate sensitivity and high false alarm rates.

The rest of this paper is organized as follows. In the next section we describe our techniques to learn normal patterns and alert on anomalous patterns. We introduce methods to detect suspicious traffic rates, suspicious new flows or the absence of expected flows, and changes in node fan-in or fan-out patterns. The development is based on statistical learning, which enables us to quantify a priori the expected false positive rate for the monitoring apparatus. This is followed by a description of our test environment, built around a commercially available Distributed Control System (DCS) communicating with a number of emulated field devices in a virtual machine environment. Next we describe characterization of normal and abnormal flows for system startup, steady state operation, nonmalicious failures of various components, and a variety of attacks. The attacks include probes, denials of service, and attempts to introduce rogue traffic. We then present experimental results to validate the usefulness of the proposed techniques. We conclude with a summary and suggestions for future work.

II. COMMUNICATION PATTERN ANOMALY DETECTION IN PCS

Process Control Systems (PCS) are often characterized by fairly regular communication patterns between master and slave units, and a fairly static address space. This can be exploited in the form of a compact ruleset that alerts to violations of expected communication patterns, presented in our earlier work [1]. Here we extend the concept to effectively learn normal flows and alert on statistical exceptions to the learned norms. This paper describes two anomaly detection techniques: pattern-based anomaly detection for monitoring the patterns of hosts with which each host communicates, and flow-based anomaly detection for monitoring the traffic patterns for individual network flows.

III. PATTERN-BASED ANOMALY DETECTION

We analyzed data traces for normal, anomalous, and attack conditions using an adaptation of the pattern anomaly detection technique of [4]. This method examines patterns in a stream of observations via a version of competitive learning [2,3]. A pattern is a vector of feature values relevant to a particular implementation; here we applied the technique to patterns formed from source and destination IP addresses and destination port. We considered that source port would typically be ephemeral (that is, system assigned) and therefore not useful, but destination port is often bound to a particular service.

Patterns are evaluated against an initially empty pattern library. If a pattern matches an existing pattern, according to some similarity function returning a value above a specified similarity threshold, then the best-matching library pattern "wins". In the (typical) case that the match is not exact, the winning library pattern is slightly adapted in the direction of the new pattern, where the degree of adaptation is based on how many previous instances of the pattern have been observed. The historical, aged number of instances also provides an estimate of the probability with which this pattern is observed.

Algorithm to pick winner :

Find K s.t.

$Sim(X,E_K) \geq Sim(X,E_k) \forall k$

X = observed pattern

E_k = kth pattern exemplar in library

If $Sim(X,E_K) \geq T_{match}, E_K$ is the winner

Else insert X into the library of pattern exemplars

T_{match} = Minimum match threshold

Adaptive modification of the winning pattern:

$$E_K \leftarrow \frac{1}{n_K + 1}(n_K E_K + X)$$

n_K = Historical (possibly aged) count of observances of E_K

An anomaly is generated when the tail probability (the historical probability of the winning pattern plus that of all patterns in the library with equal or lower probability) is below a specified anomaly threshold.

$\Pr(E_K)$ = Historical probability of pattern K

$$= \frac{n_K}{\sum_k n_k}$$

$Tail\_\Pr(E_K)$ = Historical tail probability of pattern K

$$= \sum_{\Pr(E_k) \geq \Pr(E_j)} \Pr(E_j)$$

If $Tail\_\Pr(E_K) \leq T_{alert}$, generate alert

T_{alert} = alert threshold

A periodic library update procedure consolidates similar patterns and prunes rare patterns and feature values. In practice, the heuristics used in update prevent state space explosion in numerous applications of the approach.

An attractive characteristic of this method is that it can adaptively learn multiple patterns of normal and abnormal activity (we refer to these as *modes*), whereas many other anomaly detection techniques use a two-class learning approach. In particular, the system does not require attack-free training data. Also, the use of empirical probability as the anomaly threshold allows reasonable a priori expectations on the false positive rate.

IV. FLOW-BASED ANOMALY DETECTION

We maintain a database of active and historical flow records observed in the PCS. A flow record is generated or incremented as packets are observed. As flow records are "touched" by packet traffic, they are evaluated against learned historical norms. In addition, there is a periodic global update where the flow records since the last global update are folded into historical statistical profiles. We are interested in anomalies such as observation of a new flow (with some estimate of how unlikely it is to observe a new flow), significant change in the rate (packet inter-arrival time or data volume) of a flow, and absence of an expected flow. For indexing purposes, the historical and current flow tables can be indexed by any suitably efficient scheme (for example, by some hash of source and destination).

The derivation is provided in the context of PCS, where the number of inter-communicating nodes and their addresses is relatively static, and the number of system services relatively small. The concepts can be extended to general network flows, and offer potential for monitoring quality of service (QOS) and detecting flooding-based attacks.

Packets in PCS are typically generated in a polling fashion where a master polls a number of slaves for data. Under some conditions, slaves may initiate flows to notify the master of an exception. The global database update interval should be long relative to the PCS polling interval. Notionally, we expect a reasonable tradeoff between stable statistics in a global update interval, and ability to alert in a timely fashion is probably obtained with a global update interval on the order of 100 to 1000 times the interval of the most frequent expected flow (for example, the MODBUS polling interval).

The algorithm is presented as a detector of anomalous flows, but can be easily adapted to specific flows (for example, specific to particular MODBUS function codes).

A flow record (FR) has the following elements:

FR = {Source, Dest, Tlast, Packets, Avg(NumBytes), Var(NumBytes), Avg(DT), Var(DT), Score}

- Source: source IP and port of the flow
- Dest: destination IP and port of the flow
- Tlast: time of the last packet for this flow
- Packets: number of packets in this flow since the last database update
- Avg(NumBytes): average number of bytes per packet in this flow since the last database update
- Var(NumBytes): variance of bytes per packet
- Avg(DT): mean packet inter-arrival time (DT) since last global update
- Var(DT): variance of DT
- Score: anomaly score for this record

The historical record (HR) corresponding to the same flow has the following elements:

HR = {Source, Dest, UpdateTime, HPackets, HAvg(NumBytes), HVar(NumBytes), HAvg(DT), HVar(DT)}

- Source and Dest are defined as in FR
- UpdateTime: time of last global update
- HPackets: historical aged packet count
- HAvg(NumBytes): historical average bytes per packet
- HVar(NumBytes): variance of bytes per packet
- HAvg(DT): historical average inter-arrival time
- HVar(DT): historical variance of inter-arrival time

We may relax the definition of flow to not require the same source port, as this is likely to be system assigned and ephemeral. The destination port is more likely bound to a specific system service.

A. Updating and Scoring Flow Records

We discuss the update and score procedure when we observe a packet for which there is a historical flow. In the case that there is no active flow, we allocate a new FR, set the Packets field equal to 1 and NumBytes equal to the observed packet size, TLast equal to the time of the packet, and skip the scoring. The variance calculations require at least two packets for the flow.

If there is no HR corresponding to the flow, the FR is updated normally as packets are observed, but the scoring procedure is skipped since scoring requires historical data. At the time of the next global database update, the HR is populated with the contents of the accumulated FR.

New Flow Alert: We would probably alert in the case of a flow with no corresponding HR, or alert if we see a new flow after some configurable number of global updates, or when we have observed so many flows that we consider a new flow unlikely (say, when the count of HR flows has not changed or is over some number).

Otherwise, when a packet is observed for a source-destination pair, the corresponding flow record is updated as follows:

Packets = Packets+1;

DT = current time – Tlast;

Means and variances for DT and bytes per packet in the flow records should be computed using online algorithms (http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#III._On-line_algorithm n.d.)

Note that this implies that the flow record must be expanded to include intermediate sums of squares for bytes and inter-arrival time. Also note that variances are defined only for FR records with a packet count of two or more.

The bytes per packet and inter-arrival time measures are then scored relative to the corresponding historical quantities. At present, this is based on a T test.

When the historical packet count is large (or has reached its asymptotic value), the T statistic is defined as follows (where X is replaced by packet length or DT):

$$T_X = \frac{Avg(X) - HAvg(X)}{\sqrt{Var(X)/Packets}};$$

After the scoring step, Tlast is updated to the time of the current packet.

If we treat the bytes per packet and DT as independent, we can obtain a combined score as follows:

$$Score = \sqrt{T_{NumBytes}^{\ 2} + T_{DT}^{\ 2}};$$

This has the advantage of giving one score per flow as it is touched by a packet, but loses the sense of whether the score was extremely high or low. It may be that in any case extremes on the low side are better detected at the global update interval (for example, we observe that the above will never allocate or score an expected flow that is not observed in some global update interval).

Anomalous Flow Alert: The above score exceeds some threshold.

Note that a packet updates only one flow record. The global update procedure modifies all flows.

B. *Global Update*

Global update is defined for all flows for which there is either an FR (new flow observed in the most recent global update interval), an HR (a flow was historically observed but was not observed in the most recent global update interval), or both (ideally the most common condition).

New flow alert: This is issued in the case of an FR with no corresponding HR, and was discussed in the previous section.

Missing flow alert: This is issued in the case of an HR with no corresponding FR, or an FR with 0 packets. We can either issue the alert unconditionally or score the flow as having 0 bytes and DT equal to the global update time interval. The latter approach is less likely to alert on flows that are valid but rare, since the distribution for DT would consider long inter-arrival times more normal.

Aging: We may age the HR database by multiplying with some constant AGE $\leqslant$ 1.0 (using 1.0 turns off aging). Aging allows the algorithm to adapt to changing environments.

The pseudo-code for the update equations is given below (once again, substitute NumBytes and DT for X):

// Aging packet count

$HPackets = HPackets * AGE;$

// Update averages and variances

$$HAvg(X) = \frac{HPackets * HAvg(X) + Packets * Avg(X)}{HPackets + Packets};$$

$$HVar(X) = \frac{HPackets * HVar(X) + Packets * Var(X)}{HPackets + Packets};$$

// Reinitialize FR records for the next interval

// Note that we do not reset TLast, so DT could

// span several global updates, which might be

// true for every rare flows

$Packets = 0;$

$Avg(NumBytes) = 0;$

$Var(NumBytes) = 0;$

$Avg(DT) = 0;$

$Var(DT) = 0;$

V. Test Environment

The test environment is based on a Distributed Control System (DCS) from Invensys Process Systems, IA series (http://www.ips.invensys.com/en/products/autocontrols/Pages/DistributedControl-IASeries-P018.aspx). The key elements of this system are

- An application workstation (AW) for configuration, visualization, and control. This is dual homed with a connection to a control LAN as well as an external interface.
- A control LAN based on a redundant pair of Enterasys switches (optical Ethernet).
- An Invensys Field Control Processor (FCP) module.
- A field bus that connects the FCP to (presently) two Ethernet Field Bus Modules (FBM).
- A field LAN connecting the FBM, simulated MODBUS devices (MODBUS simulators from modbustools.com and Calta) running in virtual machines.
- The monitoring system connects to the control LAN (AW, switches, and FCP) and separately to the field LAN (FBM and devices).
- Interfaces between the monitoring system and the ArcSight Security Information and Event Management (SIEM) platform.

The test environment system is shown in Figure 1. The protocol on the control LAN, between the AW and the FCP, is proprietary. The protocol on the field LAN, between the FBM and field devices, can be any of a number of common industrial protocols. For the present analysis, we have chosen MODBUS.

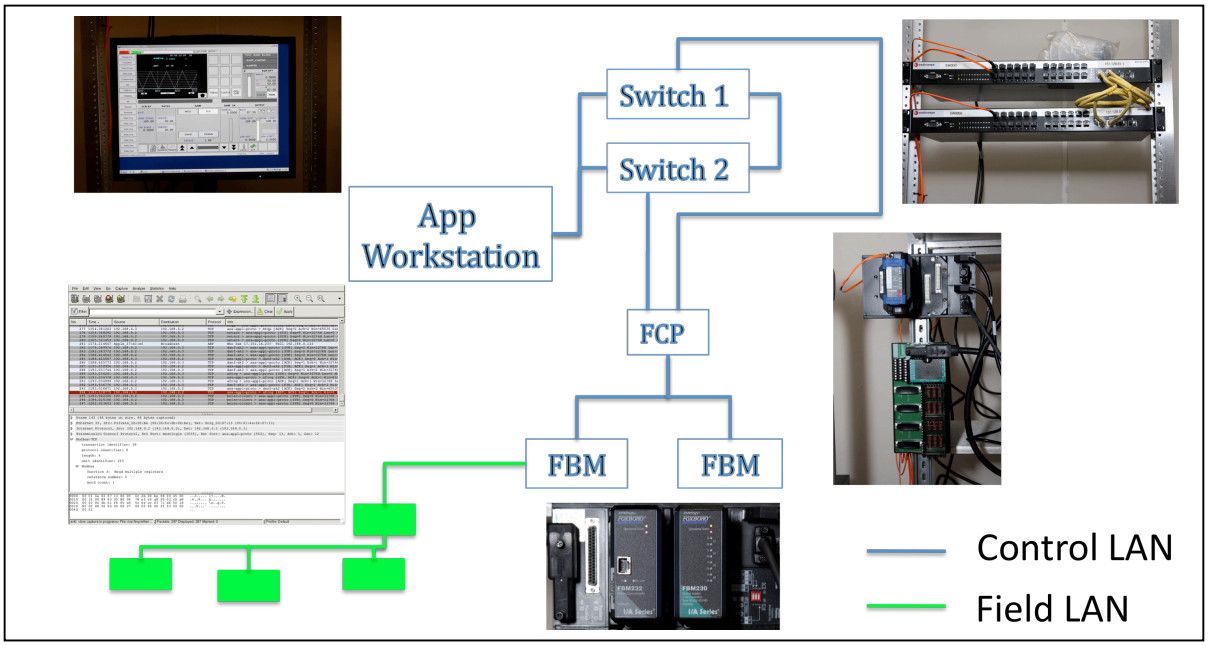

Figure 1: Schematic of DCS Testbed Environment

VI. Normal, Anomalous, and Attack Flows

We establish a baseline of normal flows by operating the system under various normal and anomalous (but not malicious) operating conditions. These conditions are representative of startup, steady-state operation, visualization of process variables, and change of process variables.

On the Invensys IA system, at startup the FCP obtains a bootp file from the AW. We collected traffic from the control LAN under normal startup as well as starting up the FCP with the AW powered down, the latter to allow us to obtain a traffic trace for the FCP's efforts to obtain the boot file.

We characterize normal flows on the field network as normal MODBUS traffic between the FBM and the emulated MODBUS PLCs. MODBUS is a master-slave protocol wherein the master polls for process values from the slaves (read) or resets continuous or logical process variables on the slaves. We also attempt to observe flows for conditions other than normal operation but not necessarily the result of malicious activity. For example, we collect data as well in the case of lost connectivity to a device or loss of the FBM itself. The latter condition effectively disconnects the field devices from the DCS, unless redundancy is built in. Process values from the field devices are communicated back to the AW through the FCP and control network.

To date, our attacks have focused on the field network, which is likely to be more exposed and therefore more vulnerable in typical operational settings. Generally, we observe that attacks of sufficient traffic volume to impede normal MODBUS communications are observed at the AW as lost connectivity to one or more MODBUS devices. Less intense attacks, and attacks involving insertion of rogue devices on the field network, are generally visible only by monitoring the field network.

VII. Experimental Validation

We first describe experimental results for evaluating pattern-based anomaly detection. Then we describe the experimental results for evaluating flow-based anomaly detection.

A. Pattern-based Anomaly Detection

We have conducted experiments to examine the usefulness of the pattern anomaly detection technique applied to IP-address-level communication. Specifically, in the experiments, for every IP address observed in the field network, we monitored the set of IP addresses with which it communicated for every time period. At the end of every time period, these communication patterns were compared with those in the pattern libraries to detect anomalous patterns and to update the pattern libraries.

1) General Experimental Setup

In the experiments, the FBM acts as the MODBUS-TCP client that periodically retrieves data from a set of (emulated) MODBUS-TCP servers, developed by Witte Software (www.modbustools.com). Another host is attached to the field network playing the role of the adversary. To generate the datasets for our experiments, we collected TCP traffic using tcpdump (www.tcpdump.org) on the field network. Moreover, we extracted the relevant fields (e.g., timestamps and IP addresses) from the packet traces, and used them as inputs for the pattern anomaly detection algorithm.

2) Normal Traffic

To evaluate the false alarm rate of the detection algorithm, we used three datasets pertaining to normal usage scenarios, and the algorithm did not report any anomalies. The scenarios include the followings

(1) FBM reads the same amount of data from all MODBUS servers at the same frequency (1 second);
(2) FBM reads the same amount of data from the MODBUS servers at different frequencies (0.5, 1, 5, and 10 seconds)
(3) FBM reads different amounts and types of data from the MODBUS servers
(4) Using two MODBUS clients (including the FBM and an (emulated) MODBUS client running on a virtual machine) to query the MODBUS servers at different frequencies (1 and 5 seconds, respectively)

In analyzing the communication patterns of these scenarios, we used the following parameters for the algorithm described in Section III. We set T_{match} to 0.6, T_{alert} to 0.7, and period length to 30 seconds. These parameters are quite "elastic" in that there exists a wide range for their values that gave the same outcome, thanks to the regularity of the network traffic patterns. For example, setting T_{match} to 0.2 or 0.9 does not change the results. Moreover, for every IP address pertaining to the MODBUS servers, the FBM, or the additional MODBUS client, the number of communication patterns is one.

3) Anomalous Traffic

To evaluate the ability of the algorithm to detect anomalous events, we used several datasets pertaining to potentially malicious events, and the algorithm reported anomalies in all those instances. These malicious events include

(1) Performing scans against the FBM or a MODBUS server using nmap (www.nmap.org)
(2) Performing network vulnerability analysis against the FBM or a MODBUS server using Nessus (www.nessus.org)
(3) Running a MODBUS client on a host and modifying MODBUS data points on a MODBUS server

In these datasets, we ran the experiment to generate about 30 minutes of normal traffic to let the detection module learn the normal traffic patterns before performing the malicious event.

4) Nmap Scans

To better illustrate the experimental setup and results, we describe the nmap scanning experiment in more detail.

In this experiment, we generated the background network traffic by having the FBM retrieve data from the four MODBUS servers at 1 second frequency (i.e., the same pattern as the normal traffic profile (1) described earlier).

After about 30 minutes, we used a network scanning tool, called nmap, to perform port scans against all TCP ports of the FBM and a MODBUS server respectively. Specifically, we ran nmap with the intense scan, all TCP ports profile, using the command
nmap –PE –v –p1-65535 –PA21,23,80,3389 –A –v –T4
against the IP address of the FBM or of a MODBUS server.

Using the similarity threshold of 0.6, anomaly threshold of 0.7, and time period of 30 seconds, the pattern anomaly detection technique detected the nmap with an anomaly score of 0.968 at the end of the first period when nmap was started. Moreover, for the nmap run against the FBM, two patterns were reported for the FBM, one pertaining to the normal traffic pattern, and one for the pattern in which there was substantial traffic between the nmap host and the FBM. In other words, the system learned distinct patterns for the normal and attack traffic, which permits labeling the latter pattern as malicious even if an adversary tries to train the system to consider the pattern normal from a threshold probability standpoint (concept drift).

The pattern set pertaining to the FBM (a MODBUS client) is as follows:

Pattern 1 (normal polling of MODBUS Servers):
MODBUS server 1: Prob 0.250
MODBUS server 2: Prob 0.250
MODBUS server 3: Prob 0.250
MODBUS server 4: Prob 0.250

Pattern 2 (attack pattern):
MODBUS server 1: Prob 0.001
MODBUS server 2: Prob 0.002
MODBUS server 3: Prob 0.001
MODBUS server 4: Prob 0.001
Nmap host: Prob 0.994

The numbers shown in the patterns correspond to the probability distribution for the number of packets exchanged between the FBM and the hosts whose names appear next to the numbers. In other words, Pattern 1 corresponds to the

pattern that the number of packets exchanged between the FBM and the four MODBUS servers is essentially the same. Moreover, Pattern 2 corresponds to the pattern that the network traffic pertaining to the FBM is dominated by the nmap network scan traffic.

For the nmap experiment pertaining to a MODBUS server, we obtained a similar result, with the anomaly score of 0.968 and two traffic patterns corresponding to the MODBUS server, one for normal traffic involving the FBM and the MODBUS server, and one corresponding to the nmap traffic and the normal traffic.

We have experimented with different values, ranging from 0.1 to 0.9, for the similarity threshold and the anomaly threshold, and obtained essentially the same result. We plan to experiment with more stealthy scanning techniques in the future.

B. *Flow-based Anomaly Detection*

To evaluate the usefulness of the flow-based anomaly detection technique of Section IV, we developed a software module for performing offline flow-based anomaly detection. The implementation defines a flow in terms of its source IP address, destination IP address, and destination port. Moreover, flows are unidirectional. In other words, an established TCP connection consists of two flows, one in each direction. The detection module can detect new flows and missing flows by comparing the flow records for the latest period and those in the historical flow database. To detect anomalous flows based on packet length and inter-packet arrival time, we use a simple heuristic to determine if the historical flow records may give a good approximation for the population means for packet length and inter-packet arrival time—using flow records in the historical database for anomalous flow detection only after we have observed the flow for more than a specified number of periods (e.g., 30). Moreover, we selected the probability threshold for detecting anomalies as 0.001. In other words, the probability that a normal flow is flagged as anomalous is less than 0.1%.

We used several packet traces collected in our DCS testbed environment to evaluate the effectiveness of the flow-based technique. In the first experiment, we significantly increased the flow rate of a MODBUS connection to simulate a surge in requests for a MODBUS server, which may be caused, for example, by a flooding-based denial-of-service attack. In the second experiment, we significantly reduced the flow rate of a MODBUS connection, which may indicate a system degradation problem at the MODBUS client. In the third experiment, we used a MODBUS-TCP scanner developed by Mark Bristow (http://code.google.com/p/modscan) against a MODBUS server; the tool attempts to discover the unit identifiers managed by MODBUS-TCP servers.

1) *Increasing Flow Rate*

In this experiment, we employed two MODBUS clients to fetch data from the MODBUS servers in the testbed. The FBM queried the four MODBUS servers once per second, and this rate stayed constant throughout the experiment. A second MODBUS client, which had a different IP address than that of the FBM, initially queried a MODBUS server once every 10 seconds. After 2.5 hours, we changed the MODBUS request rate to once per second, and let the experiment run for another 1.5 hours.

With a period length of 180 seconds, at the end of the first period after the change of the request rate, the T statistic score for DT of the flow from the second MODBUS client to the MODBUS server was -11.13. For the sample size of 106 (i.e., the number of packets in the flow observed in that period), the absolute value of the T statistic should be less than 3.39 with 99.9% probability. As a result, the flow was flagged as anomalous. The T statistic for DT of the flow became -232.80 at the end of the next period, when more anomalous traffic was observed, which also triggered the anomaly detection algorithm to generate an alert. As time went on, the historical record for the flow was updated with data pertaining to the higher MODBUS request rate, and the T statistic for DT gradually decreased and became -28.63 at the end of the experiment.

2) *Decreasing Flow Rate*

This experiment was similar to the one pertaining to increasing flow rate. The main difference was that we decrease the MODBUS request rate from once every second to once every two seconds, as opposed to increasing the request rate.

With a period length of 60 seconds, at the end of the first period after the change of the request rate, the T statistic score for DT of the flow from the second MODBUS client to the MODBUS server was 3.65. For the sample size of 54, the absolute value of the T statistic should be less than 3.49 with 99.9% probability. As a result, the flow was detected as anomalous.

3) *Modscan*

In this experiment, the normal traffic profile is similar to that of the flow rate experiments, except that we performed a MODBUS unit identifier scan from the MODBUS client host against the MODBUS server using the modscan tool. The scan, lasting for several minutes, involved sending a number of MODBUS requests to the MODBUS server with different unit identifiers for determining the valid identifiers based on the responses. Every MODBUS request used in the scan involved establishing a new TCP connection, which affected not only the DT statistic, but also the packet length statistic for the flow.

With a period length of 60 seconds, at the end of the first period after modscan was started, the T statistic score for DT of the flow from the MODBUS client host to the MODBUS server was -12.98. For the sample size of 247, the absolute value of the T statistic should be less than 3.373 with 99.9% probability. Thus, the flow was detected as anomalous. At the end of the next period, the T statistic scores for DT and for packet length were -75.16 and -8.43, respectively—both greater than the threshold needed for anomaly detection.

The results of the flow anomaly detection indicate that the technique is able to detect anomalous flows effectively, but it is somewhat more subject than the pattern anomaly approach to concept drift.

VIII. Conclusion and Future Work

We investigated the usefulness of communication pattern anomaly detection for process control networks. Specifically, we conducted experiments to evaluate two anomaly detection techniques, namely, pattern-based detection for communication patterns among hosts, and flow-based detection for traffic patterns for individual flows. Our initial experimental results are encouraging. The absence of anomaly reports in the normal traffic supports our hypothesis that anomaly-based detection is feasible in process control networks at much lower false alarm rates than in general enterprise systems. These techniques were able to detect some basic attacks launched against the MODBUS servers in our DCS testbed. As for future work, we plan to develop real-time intrusion detection sensors based on these techniques, and to test them in more realistic environments.

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-FC26-07NT43314.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

[1] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using Model-based Intrusion Detection for SCADA Networks", Proceedings of the SCADA Security Scientific Symposium, D. Peterson, ed., 2007.

[2] S. Grossberg (ed.), "Neural Networks and Natural Intelligence," MIT Press, 1988.

[3] S. Kohonen, "Self-Organizing Maps", 3rd Edition, Springer, 2001.

[4] A. Valdes, "Detecting Novel Scans Through Pattern Anomaly Detection", Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX III), Volume 1, pp. 140-151.