Chapter 6

LIGHTWEIGHT JOURNALING FOR SCADA SYSTEMS VIA EVENT CORRELATION

Antoine Lemay, Alireza Sadighian and Jose Fernandez

Abstract Industrial control systems are not immune to cyber incidents. However, the support for incident responders and forensic investigators is low. In particular, there are limited journaling capabilities for operator actions. Barring the preservation of full packet captures and operator workstation security logs, which can generate unmanageable amounts of data on production networks, it is generally not possible to attribute control events (e.g., opening a valve or operating a breaker) to individual operators. This information can be necessary to perform security investigations, especially in cases involving malicious insider activities. This chapter presents a lightweight journaling system for SCADA networks based on event correlation. By correlating network events and operating system logs, a journal is generated of all Modbus protocol write events along with the usernames of the operators who performed the actions. The journal is much more compact than a full packet capture, achieving compression ratios of around 570 to 1 in conservative conditions and more than 2,000 to 1 in typical operating conditions, allowing for the preservation of valuable information for security investigations.

Keywords: SCADA networks, network forensics, journaling, event correlation

1. Introduction

The number of cyber incidents has been rising in recent years. Industrial control networks (also referred to as supervisory control and data acquisition (SCADA) networks) are not immune to cyber threats; examples are the Havex malware [10] and a serious incident at a German steel mill [1]. This trend underscores the need for better incident response capabilities in industrial control or SCADA networks.

M. Rice and S. Shenoi (Eds.): Critical Infrastructure Protection X, IFIP AICT 485, pp. 99–115, 2016.
DOI: 10.1007/978-3-319-48737-3_6

One of the first instincts of an incident responder is to review logs to find clues about the incident. However, this is usually not possible in SCADA networks. In the vast majority of these networks, the control systems do not support the journaling of relevant security information. This lack of journaling is critical from a security perspective. In a SCADA system, the controllers perform all the interactions with the physical network. An attacker who wishes to create a physical impact must interact with the controllers. Yet, there is no convenient way to track these attacker interactions.

Full packet captures can be used to track most interactions with controllers. Typical controllers do not possess any keyboards or user displays and, unless an individual is performing maintenance using physical access, the controllers are usually only accessed remotely via the control network. In this sense, full packet captures record the interactions with the controllers. However, they also contain large volumes of irrelevant network activity, preventing their storage for extended periods of time.

As an example, the limited test network used in this research generates traffic in the order of gigabytes per month. In a production network, with many more controllers and field devices, the network traffic would be orders of magnitude higher. As such, there is a need to preserve summary information that would enable incident responders to gather key information about the interactions with controllers. This chapter describes an event correlation approach that creates a log of interactions between operators and controllers by combining data from network sensors and data from security logs in operator workstations.

2. Background

This section presents background information about SCADA systems. In particular, it describes the architecture and operation of SCADA networks that use the popular Modbus protocol.

Many SCADA networks are organized around the Purdue Enterprise Reference Architecture [15]. In this architecture, Level 0 is the physical process. Intelligent field devices, namely, sensors, actuators and their controllers, are located in Level 1. Higher level controls, such as SCADA systems, are in Level 2. All Level 1 equipment is grouped together on the same local-area network (plant local-area network). Similarly, all Level 2 equipment is grouped in the control center local-area network.

Figure 1 presents the architecture of a Modbus system. In the architecture, each operator workstation hosts a SCADA program. The SCADA program comprises two main components. The first component is the master terminal unit (MTU), which is responsible for maintaining information about the state of the physical process in the operator workstation. In Modbus, this is accomplished by continually polling each controller to request updates on the states of all the field devices attached to the controller. Every few seconds, the exact time being defined by the polling interval configuration option, the master terminal unit sends a Modbus read packet to each controller. The controller

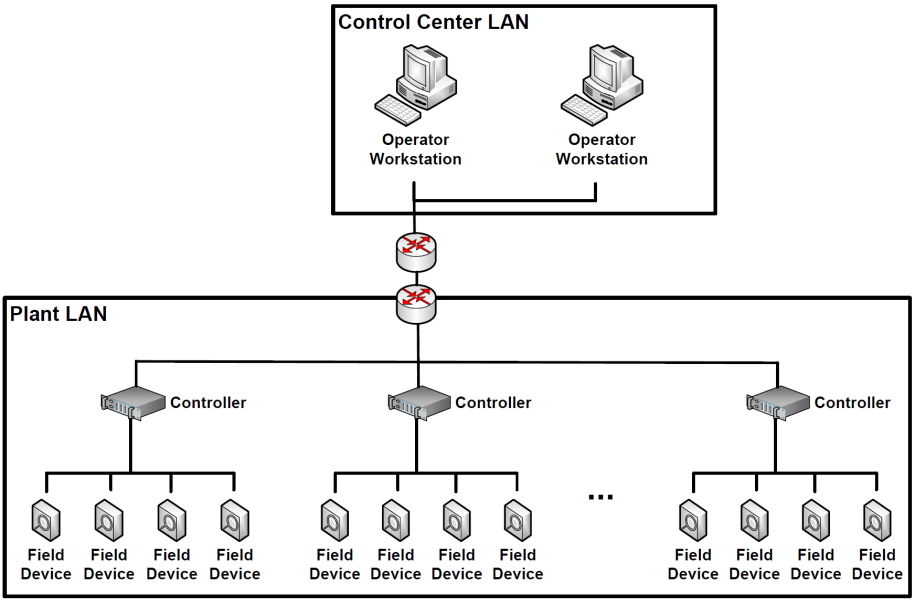

Figure 1. Modbus system architecture.

responds with a read response packet containing the values of all the requested registers. This process is referred to as polling traffic in the rest of this chapter.

The second component residing in the operator workstation is the human-machine interface (HMI). This component provides the operator with a visualization of the state of the physical process based on the most recent values gathered by the master terminal unit. It also provides the operator with a graphical interface for altering the state of the system. As soon as the operator performs the action, the master terminal unit sends a Modbus write request to the controller that supervises the relevant field device. The controller then alters the state of the field device and returns a write response packet. This is referred to as command traffic in the rest of this chapter.

In a security context, polling traffic is of little interest because it is the result of an automated process and has little impact on the well-being of the physical process. However, command traffic is (usually) the result of deliberate human actions and has an immediate impact on the state of the physical system. In this sense, command traffic is very relevant in a security context. Unfortunately, SCADA software does not automatically log actions performed by human operators. Therefore, if this information must be retained for investigative purposes, it is necessary to devise a method for gathering and preserving the information.

3. Journaling Use Cases

One of the most important use cases of journal examination is in the incident response context. The presence of a journal of operator actions enables incident investigators to correlate events in the journal with effects on the physical system. As an example, consider an investigation of an overflow caused by the unexpected opening of a valve. It would be possible to examine the journal log to see if an operator sent commands around that time and question the operator about the actions he performed around the time of the incident. This capability is particularly useful in cases involving human error or malicious acts by disgruntled employees.

Developing a forensics capability in an organization not only helps with incident response, but also with incident prevention. Much like the presence of security cameras, public knowledge of the forensics capability can actively deter malicious activities because of the threat of being caught. The same effect is applicable to limiting human error because of the ability to implement negative incentives for carelessness.

A second use case of is the detection of properly-formatted malicious commands. In normal control system operation, commands are sent from a limited set of machines, typically operator workstations. Only the logs from these machines can be correlated with network events. The flipside is that the correlation would fail if the machine used to send commands was not an operator workstation. It is unlikely that local security event logs would be collected if an attacker has established a presence on a machine other than an operator workstation (e.g., a controller) or has introduced a new machine in the network. Thus, correlation failures are highly indicative of malicious activity and can be used to generate alerts.

A third use case is the detection of malicious software or remote compromise in an operator workstation. A situation where a Windows Management Instrumentation (WMI) query is successfully completed (i.e. the Modbus command comes from a legitimate workstation) but the query fails to find an active operator on the workstation would imply that the Modbus command was sent by a remote or automated process. In the majority of cases, this would be unusual because a human operator is expected to be involved in any system modification and a modification made by a remote user or process may be an indicator of malicious activity.

4. Event Correlation

Considerable research has been conducted in the area of event correlation. However, most of the work is focused on enhancing intrusion detection systems. For example, Valeur et al. [14] have proposed an alert correlation workflow comprising ten steps, namely normalization, pre-processing, alert fusion, alert verification, thread reconstruction, attack session reconstruction, focus recognition, multi-step correlation, impact analysis and prioritization, to correlate the alerts of multiple intrusion detection systems. However, their approach, which

focuses on alarm generation, is ill-suited for this research, which concentrates on legitimate traffic.

Sadighian et al. [12] have created an ontology-based alert correlation framework that, like the approach of Valeur et al. [14], could supplement this research, but this would require significant customization because large portions of the framework created to discover attacks would not apply. Saad et al. [11] have proposed a hybrid alert correlation approach using semantic analysis and an intrusion ontology to reconstruct attack scenarios, but the approach is also difficult to adapt to the current context. Specifically, in this context, the observed malicious activity is not the complex multi-step attacks for which the approach of Saad et al. [11] is optimized. Finally, Ficco [5] has developed a hybrid, hierarchical event correlation approach for detecting complex attacks; unfortunately, this approach does not suit the goals of this research.

Even if the approaches described above could be applied, event correlation techniques developed for intrusion detection suffer from a number of other drawbacks. Yusof et al. [17] have analyzed alert correlation techniques and list many of the drawbacks. In addition to alert flooding and false alerts, scalability and an inability of understand contextual information are identified as limitations. However, these limitations are not expected to impact SCADA networks, where the context seldom changes and the scale is limited compared with traditional information technology networks. In this sense, event correlation appears to be appropriate for merging information from network events and operating system events.

The classification of correlation approaches provides useful research insights. Cuppens and Miege [4] have classified attack reconstruction approaches into two categories: (i) implicit alarm correlation; and (ii) explicit alarm correlation. Implicit alarm correlation is based on employing machine learning and data mining techniques (see, e.g., [7]). Explicit alarm correlation relies on the ability of security administrators to express logical and temporal relationships between alerts in order to detect complex multi-step attacks (see, e.g., [3]).

In the SCADA network context, the relationships between events are dictated by the heavily-constrained operational context. For example, an operator can only interact with the system via a human-machine interface. As such, explicit alarm correlation appears to be a practical choice for the operational technology (OT) journaling approach described in this chapter.

5. Journal Generation Approach

This section details the general approach for generating the operational technology journal. The section begins by describing the general architecture and proceeds to present the correlation methodology.

5.1 General Architecture

In a typical SCADA network, operators are only allowed to interact with controllers through specific channels. Specifically, they use operator workstations

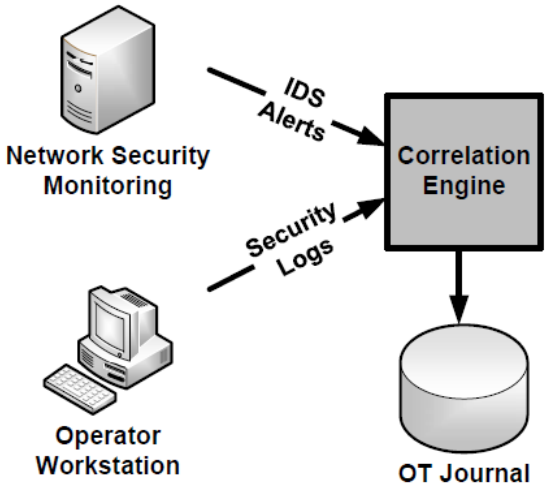

Figure 2. General architecture.

that host the proprietary software required to interact with the controllers. The operator workstations usually run commercial off-the-shelf operating systems. As such, the workstations routinely generate security log entries during their operation. The security log entries allow the identification of operators based on their usernames.

While an operator workstation creates a log entry when a user logs on, there is limited visibility of the user actions beyond the login entry. Typically, the program that is used to perform operations does not keep records of the actions performed by an operator. Thus, if an actuator is activated and five operators are present in the control room, there is no way to find out which operator performed the action by inspecting the logs on the operator workstations alone.

Similarly, it is not possible to identify an operator by examining network traces. The only information derived from network traces is based on the data contained in the network packets. Typically, this includes information about the source and destination (IP or MAC) addresses and information in the packet payloads. Because of how SCADA protocols operate, this information is usually sufficient to find out exactly which action occurred on a controller and to identify the operator workstation that sent the request; however, the identity of the operator cannot be determined.

To combine the available information from the security log with network monitoring traces, it is necessary to correlate the network monitoring events (packets identifying the source IP address and action performed) with the security log events from the relevant machine (identity of the logged user). Once this is done, it is not necessary to preserve the entire contents of a network trace or security log, only the result of the correlation, which is referred to as the operational technology (OT) journal. Figure 2 illustrates the general architecture.

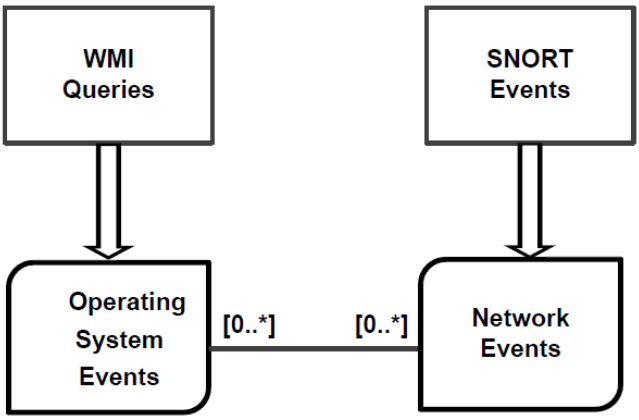

Figure 3. Relationships between concepts.

Because the operational technology journal only takes a fraction of the space required by security logs and network traces, it is possible to preserve the journal for long periods of time. The availability of the information might also meet other needs such as evidence for compliance purposes or information for performance reviews of operators.

5.2 Correlation Methodology

In order to successfully correlate events, it is necessary to find association relationships between the events in a security log and network trace. The first step in the correlation process is to take the specific events in the two sources and create abstract representations of each type of event (i.e., operating system events and network events). The components of the abstract representations are then used to establish associations between the operating system events and network events. The process is illustrated in Figure 3 for Windows Management Instrumentation (WMI) as the source of operating system events and Snort as the source of network events.

The specific components of the abstract representation used for correlation are IP address and time. To correlate the IP address, the source IP address of the network event is matched against the IP address of the operator workstation that provided the event logs. When a Modbus write event is observed in the network, the source IP address represents the workstation on which the Modbus write command was processed. As such, the operator who was logged on at the time that the event was sent should be the one who triggered the event.

To match the time, it is necessary to create a centralized log server or security information and event management (SIEM) system that would help match the timestamp of the network event with the timestamps of security events in the operating system event log. Another option is to leverage the near-real-time nature of network event generation to achieve the same results. In general, the time that an interesting network event is observed is only seconds away from

the time that the event was generated. Thus, if a live query of the operating system events could be performed as soon as a network event is observed, the responses from the operating system would be time-correlated with the network event.

6. Performance Evaluation

This section discusses the performance of the journal generation approach. It begins by describing the performance testing implementation. Next, the experiment design and performance evaluation results are presented. Finally, the results of a sensitivity analysis of the major parameters are provided.

6.1 Implementation

Two prototypes, one based on Snort and the other based on the `tshark` tool [16], were used to test the validity and performance of the proposed approach. The proof-of-concept implementation monitors Modbus network traffic and logs the usernames of the operators who send commands (i.e., interact with controllers using write commands).

The first prototype implements network monitoring as a virtual machine running the Security Onion network security monitoring image [2] in the standalone mode. The popular Snort intrusion detection system is used to parse captured network packets using Modbus preprocessors. The preprocessors enable the complete decoding of Modbus packets and the generation of alerts when specific Modbus function codes are used.

The following alerts are used in the proof-of-concept implementation:

- ```
  alert tcp any any -> HOME_NET 502
  (msg:"Modbus write_multiple_coils";
  modbus_func:write_multiple_coils; sid:9000001;rev:1;)
  ```

- ```
  alert tcp any any -> HOME_NET 502
  (msg:"Modbus write_multiple_registers";
  modbus_func:write_multiple_registers; sid:9000002;rev:1;)
  ```

- ```
  alert tcp any any -> HOME_NET 502
  (msg:"Modbus write_single_coil";
  modbus_func:write_single_coil; sid:9000003;rev:1;)
  ```

- ```
  alert tcp any any -> HOME_NET 502
  (msg:"Modbus write_single_register";
  modbus_func:write_single_register; sid:9000004;rev:1;)
  ```

Modbus uses the standard port 502. In instances where Modbus controllers do not use the standard port, 502 is replaced by the non-standard port number.

In the second prototype, the network monitoring component is implemented using `tshark` [16] running in a Windows virtual machine. As in the case of

Snort, `tshark` extracts all the packets with Modbus function codes corresponding to write commands. For each packet, the timeframe, source and destination IP addresses, exact register accessed (Modbus reference number) and value written to the register (Modbus data) are saved in a CSV file. This enables the journaling of more information than the prototype based on Snort, but it takes up more space.

The correlation engine monitors the alerts raised by both prototypes. This is implemented by a Python script. When the correlation engine detects the presence of an alert, the correlation engine extracts the IP address from the alert and immediately sends WMI queries to the appropriate IP address to obtain the name of the user who is currently logged on.

In the case of the Snort-based prototype, WMI queries are sent via a Linux WMI client invoked by the Python correlation engine script. The first query gathers the session IDs of all the active sessions on the machine that are of Type 2 (i.e., interactive):

- `"SELECT LogonId FROM Win32_LogonSession WHERE LogonType = 2"`

This represents the sessions in which the user is interacting with the machine via a graphical user interface (i.e., operator sessions).

The second query obtains the names of the users who are currently logged on along with their session IDs:

- `"SELECT * FROM Win32_LoggedOnUser"`

The usernames are then matched to the interactive sessions to find the usernames of the operators.

Similar queries are used by the `tshark`-based prototype. However, the Python WMI library [6] is used instead of a Linux WMI client.

All the operator workstations are configured to accept WMI queries. When a query arrives, the operator workstation processes the query and returns the name of the user who is currently logged on. If the query cannot be processed, it is usually an indicator of malicious activity. It could mean that the command was sent by a program or a human using a remote service instead of by a legitimate operator. Alternatively, it could mean that the command was not sent from an authorized operator workstation.

After the response from the WMI query is received, the Python correlation engine generates an operational technology journal entry comprising the timestamp from the alert, username from the WMI response, Modbus function code from the alert, destination controller from the alert and, in the case of the `tshark`-based prototype, register location(s) and data written from the alert. The journal entry is then recorded using the Python Syslog facility.

The log entry has the following format:

- `WARNING:root:['02/12-22:21:44.927339',`
  `'[1:9000003:1] Modbus write_single_coil', '192.168.1.100',`
  `'192.168.1.101', 'Alice']`

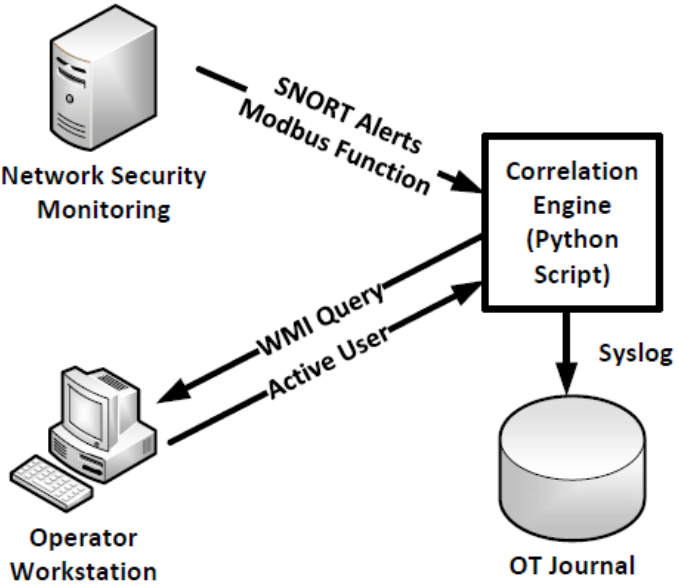

Figure 4. Proof-of-concept implementation.

In the case of the `tshark`-based prototype, the log entry has the following format:

- `WARNING:root:['Feb 26, 2015 14:56:03.503279000 Eastern Standard Time', 'WriteSingleCoil', '192.168.1.100', '192.168.1.103', '3', '00:00', 'Bob']`

The `Warning` indicator is related to the Syslog journaling level of the entry. The indicator is followed by `root`, indicating the user who was responsible for the log entry. Following this is the operational technology journal entry in brackets, which contains the timestamp, rule number and version along with the rule description, source IP address, destination IP address and, finally, the username of the operator. The `tshark`-based version adds the register number and the data written between the destination IP address entry and the operator entry. While the exact formatting of a production version would balance human readability and data compression, the suggested format is appropriate for a proof-of-concept implementation. Figure 4 presents the proof-of-concept implementation.

6.2 Performance Evaluation

The performance evaluation was performed on a test Modbus network implemented in an industrial control system sandbox [9]. The Modbus network was chosen due to the availability of a Modbus preprocessor in Snort and the availability of Modbus clients to send commands. The test network incorporated three Modbus controllers running Modbus servers developed using Modbus-

tk [8]. Each controller had four operable coils representing on/off switches. Each controller also had four binary input points replicating the states of the on/off switches and four holding registers containing values derived from the states of the switches.

The number of measurement points has a direct impact on the amount of polling traffic (irrelevant traffic from a security standpoint) in the network – the greater the number of points, the greater the amount of irrelevant traffic. Also, the greater the proportion of irrelevant traffic, the higher the compression factor. As such, it is believed that the limited amount of points do not deter from the generality of the performance evaluation because the solution performs better as the size of the network increases.

The network also contained two operator workstations with several users logged on. Each workstation ran a Modbus master terminal unit that populated a human-machine interface constructed using SCADA-BR [13]. The polling interval was ten seconds for each remote terminal unit. This value is larger than the two to five seconds used in the industry because the shorter the interval, the greater the amount of polling traffic in the network. Because polling traffic is irrelevant from a security standpoint, smaller polling intervals produce better compression results.

Additionally, on each workstation, a Python script was executed that continually sent write commands to a randomly-selected coil and then waited for a random amount of time (between 10 and 120 seconds). This traffic is interesting from the security perspective, so a shorter wait period produces a lower compression factor. However, it is important to note that operators perform actions by clicking on elements on a graphical interface and are supposed to operate these points only when changes are needed to the physical process. As such, these values represent a fairly high volume of activity in comparison with a real operator.

Finally, a Security Onion network monitoring virtual machine was launched to capture network traffic. The Security Onion machine also ran the Snort-based operational technology journal prototype. The Snort-based prototype produced log entries that were slightly smaller than the `tshark`-based prototype. This had a marginally positive effect on the compression ratio. However, it was at the expense of the quantity of information preserved. Figure 5 presents the layout of the test network.

With all the machines in place, the experiment was executed for one hour and the following metrics were recorded:

- Logs of the coils operated from each workstation (ground truth).
- Size of the full packet capture.
- List of the operational technology journal entries.
- Size of the operational technology journal.

The list of operational technology journal entries was then compared with the logs of the coils operated from each workstation. This helped validate that

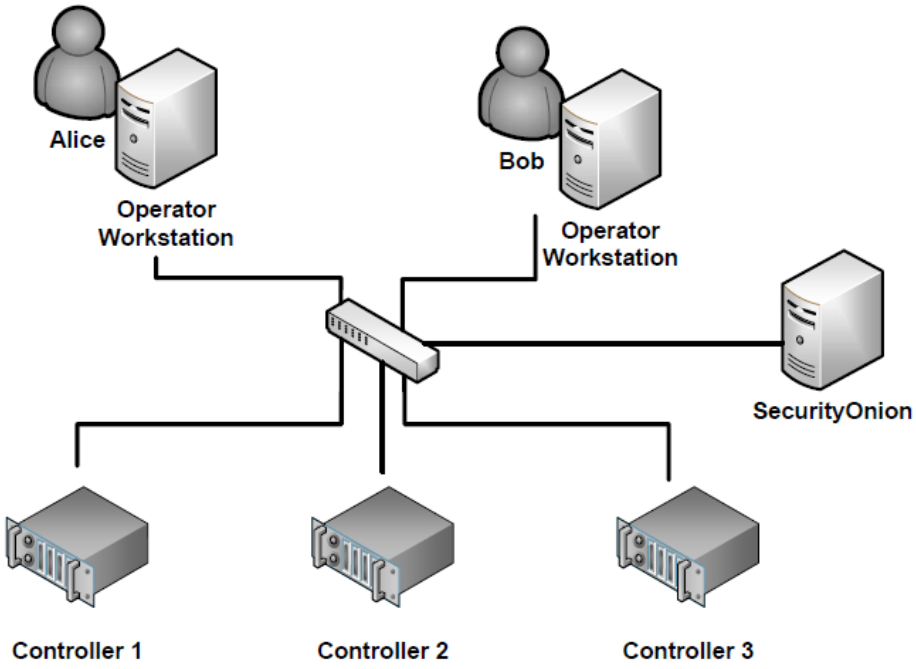

Figure 5. Test network diagram.

the approach performs according to design. Next, the size of the operational technology journal was compared with the size of the full packet capture to obtain a compression ratio. Higher compression ratios enable longer periods of storage for the operational technology journal and provide greater utility for forensics and incident response activities. As such, this was the main performance metric used in the experimental evaluation.

6.3 Experimental Results

Because the number of operator actions was generated at random, ten instances of the experiment were conducted. The raw metrics were recorded for each run. The average and standard deviation values over the ten runs were then computed. Table 1 presents the results.

The results show that the number of operations recorded is always equal to the number of operations performed. This implies that the prototype successfully records all the legitimate operations. Furthermore, the journaling functionality is implemented properly.

Over one hour of operation, an average of 12.3 KB of operational technology journal data was recorded. This yields a compression ratio of around 1:572, which is comparable with the rule-of-thumb ratio of 1:1,000 or 1:2,000 typically used for Netflow flow summary information. Since this ratio is achieved given the high level of operator activity, low polling frequency, small number

Table 1. Experimental results.

Run	Operations Performed	Operations Recorded	OT Journal Size (KB)	pcap File Size (MB)	Compression Ratio
1	95	95	12	7.0	1:582
2	98	98	12	7.0	1:583
3	102	102	13	7.2	1:553
4	94	94	12	6.9	1:572
5	92	92	12	6.8	1:570
6	99	99	12	7.1	1:588
7	105	105	13	7.3	1:562
8	99	99	12	7.0	1:585
9	95	95	12	6.8	1:570
10	105	105	13	7.2	1:555
Avg	98.4	98.4	12.3	7.0	1:572
SD	4.5	4.5	0.5	0.2	13

of points and controllers and lack of optimization of the journal format, the approach is viable in terms of long-term preservation. The choices were made to create a challenging benchmark but, even if more realistic parameters had been used, a compression ratio exceeding that achieved by Netflow would have been obtained.

With regard to preservation, even in the face of high operator activity with an average of more than 98 state alterations per hour, the volumes of the operational technology journals suggest that long-term retention would be feasible. Upon extrapolating the values obtained over one hour, a full day's worth of operational technology journal entries would be around 295 KB and an entire year's worth of entries would be less than 108 MB. In contrast, given the volume of polling traffic, preserving the full packet captures would require around 62 GB of storage. Since the polling traffic was obtained from a small experimental network, a production network would require some orders of magnitude greater than 62 GB of storage, almost certainly requiring the implementation of the proposed approach.

It is important to note that the size of the operational technology journal scales with the number of operations while packet capture scales with the numbers of controllers and points. Thus, the retention costs of the operational technology journal would be much less than the retention costs of full packet captures when scaled to the size of an enterprise.

6.4 Sensitivity Analysis

Sensitivity analysis experiments were conducted to ascertain the impact of the environmental parameters selected in the experiments. A single experimental run was conducted for each variation of the parameters.

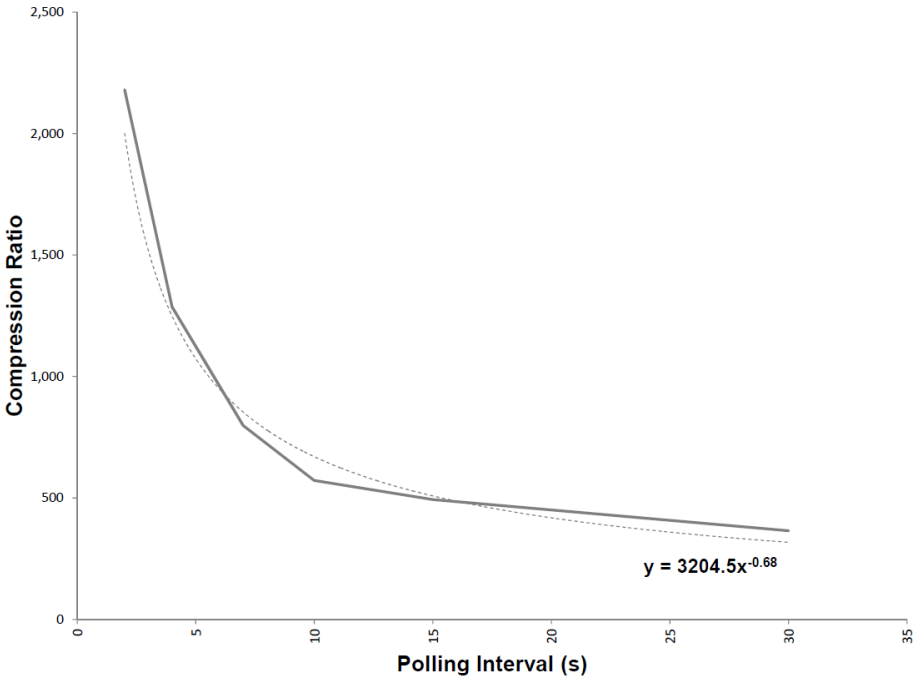

Figure 6. Impact of polling interval on compression ratio.

The first sensitivity experiment used the `tshark`-based prototype instead of the Snort-based prototype. When comparing the results against an experimental run with a similar number of operations, an increase of around 17% of the size of the operational technology journal was observed. This represents a very modest increase and suggests that the cost of preserving the additional information obtained from `tshark` is negligible compared with the standard storage size. In the case of the example discussed above, an entire year's worth of information would be 125 MB instead of 108 MB in size. It would difficult to imagine that an organization willing to store 108 MB of information would not store 125 MB, just 17 MB more information.

The second sensitivity experiment evaluated the impact of the polling interval on the compression ratio. Starting from the basic experimental layout, the polling interval between the master terminal unit and remote terminal unit was modified. A single experiment was run for intervals of 2, 4, 7, 15 and 30 seconds and the compression ratios were compared with the average compression ratio obtained in the basic experiment.

Figure 6 presents the results. As expected, the compression ratio decreases with a higher polling interval as the ratio of irrelevant traffic to relevant traffic decreases. Regression shows that the decrease follows a power-law-based function. This means that, for polling intervals commonly used in industry (i.e., two seconds), the compression ratio obtained using the proposed research

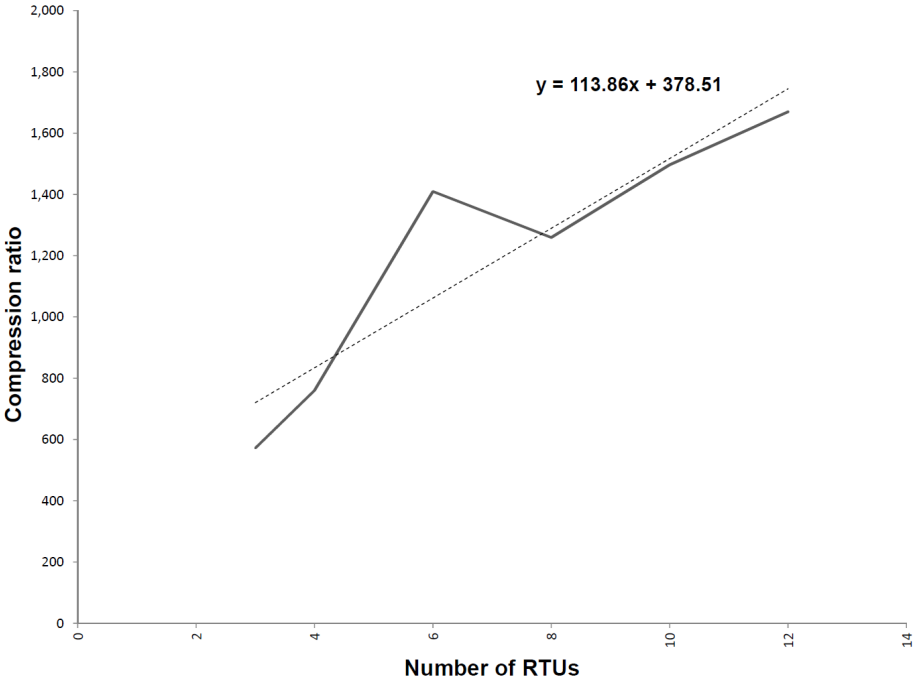

Figure 7. Impact of number of remote terminal units on compression ratio.

is comparable to the compression ratio expected for traffic record summaries (i.e., around 2,000 to 1).

The third sensitivity experiment evaluated the impact of the number of remote terminal units on the compression ratio. Starting from the basic experimental configuration, the number of remote terminal units used in the experiment was modified. A single experiment was run for 4, 6, 8 and 12 remote terminal units and the compression ratios were compared with the average compression ratio obtained in the basic experiment.

Figure 7 presents the results. As expected, adding more remote terminal units adds more irrelevant traffic and increases the compression ratio. Regression shows that the compression ratio increases linearly with the number of remote terminal units. A compression ratio comparable to traffic record summaries is expected for around fourteen remote terminal units.

Similar experiments could have been run for other design choices such as the rate of operation and the number of points per remote terminal unit. However, the current results show that the compression ratio follows the expected trends and that the design choices are sufficiently conservative to represent a lower bound on the performance of the proposed system. This performance, given more realistic parameters, may well surpass the compression ratios of traditional mechanisms such as flow summary records.

7. Conclusions

This chapter has presented an event correlation approach for creating a journal of operational events. The operational technology journal combines information obtained from network events such as Snort alerts and operating system events gathered via WMI queries to create entries of all write operations with the usernames of the operators who performed the operations. The journal is useful for incident handling, attack deterrence and even attack identification.

The main benefit of the operational technology journal is that it requires much less space than the combination of full packet captures and operating system logs. Under conservative conditions, compression ratios exceeding 570 to 1 were achieved. Indeed, the compression ratio can even surpass the 2,000 to 1 ratio usually attributed to flow summary records in realistic conditions. This enables the preservation of the operational technology journal for long periods of time for network forensic purposes.

While the correlation approach is very general, the implementation described in this chapter is limited to SCADA protocols for which parsers are available. Future research will attempt to extend the implementation to cover major proprietary protocols. In addition, research will focus on a security information and event management (SIEM) system. Finally, extending the approach to other types of events of security interest, such as uploading new configurations from engineering workstations, will also be investigated.

Acknowledgements

This research was partially funded by the Canadian Center for Security Science (CSS). In addition, the authors wish to thank the National Energy Infrastructure Test Center (NEITC) for their testing and support.

References

[1] BBC News, Hack attack causes "massive damage" at steel works, December 22, 2014.

[2] D. Burks, Security Onion Project (`github.com/Security-Onion-Solutions/security-onion`), 2016.

[3] B. Cheng and R. Tseng, A context adaptive intrusion detection system for MANET, *Computer Communications*, vol. 34(3), pp. 310–318, 2011.

[4] F. Cuppens and A. Miege, Alert correlation in a cooperative intrusion detection framework, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 202–215, 2002.

[5] M. Ficco, Security event correlation approach for cloud computing, *International Journal of High Performance Computing and Networking*, vol. 7(3), pp. 173–185, 2013.

[6] T. Golden, WMI 1.4.9 (`pypi.Python.org/pypi/WMI`), 2003.

[7] M. Hoque, M. Mukit and M. Bikas, An implementation of an intrusion detection system using a genetic algorithm, *International Journal of Network Security and its Applications*, vol. 4(2), pp. 109–120, 2012.

[8] L. Jean, `modbus_tk 0.4.3` (`pypi.python.org/pypi/modbus_tk/0.4.3`), 2014.

[9] A. Lemay, J. Fernandez and S. Knight, An isolated virtual cluster for SCADA network security research, *Proceedings of the First International Symposium for ICS and SCADA Cyber Security Research*, pp. 88–96, 2013.

[10] NETRESEC, Full Disclosure of Havex Trojans, Orsundsbro, Sweden (`www.netresec.com/?page=Blog&month=2014-10&post=Full-Disclosure-of-Havex-Trojans`), 2014.

[11] S. Saad and I. Traore, Extracting attack scenarios using intrusion semantics, *Proceedings of the Fifth International Symposium on the Foundations and Practice of Security*, pp. 278–292, 2013.

[12] A. Sadighian, J. Fernandez, A. Lemay and S. Zargar, ONTIDS: A highly flexible context-aware and ontology-based alert correlation framework, *Proceedings of the Sixth International Symposium on the Foundations and Practice of Security*, pp. 161–177, 2014.

[13] SourceForge, ScadaBR (`sourceforge.net/projects/scadabr`), 2016.

[14] F. Valeur, G. Vigna, C. Kruegel and R. Kemmerer, Comprehensive approach to intrusion detection alert correlation, *IEEE Transactions Dependable and Secure Computing*, vol. 1(3), pp. 146–169, 2004.

[15] T. Williams, The Purdue Enterprise Reference Architecture, *Computers in Industry*, vol. 24(2-3), pp. 141–158, 1994.

[16] Wireshark Foundation, `tshark` (`www.wireshark.org/docs/man-pages/tshark.html`), 2016.

[17] R. Yusof, S. Selamat and S. Sahib, Intrusion alert correlation technique analysis for heterogeneous log, *International Journal of Computer Science and Network Security*, vol. 8(9), pp. 132–138, 2008.