Detecting Integrity Attacks on Industrial Control Systems

Chad Arnold, Jonathan Butts, Krishnaprasad Thirunarayan

Chapter 1

DETECTING INTEGRITY ATTACKS ON INDUSTRIAL CONTROL SYSTEMS

Chad Arnold, Jonathan Butts and Krishnaprasad Thirunarayan

Abstract Industrial control systems monitor and control critical infrastructure assets such as the electric power grid, oil and gas pipelines, transportation systems and water treatment and supply facilities. Attacks that impact the operations of these critical assets could have devastating consequences to society. The complexity and interconnectivity of industrial control systems have introduced vulnerabilities and attack surfaces that previously did not exist. The numerous communications paths and ingress and egress points, technological diversity and strict operating requirements provide myriad opportunities for a motivated adversary. This paper investigates the detection of integrity errors in industrial control systems by correlating state values from field devices. Specifically, it considers a formulation of the classic Byzantine Generals Problem in the context of industrial control systems. The results demonstrate that leveraging physical system properties allows the inference of system states to identify integrity compromises.

Keywords: Control systems, integrity attacks, Byzantine Generals Problem

1. Introduction

On June 26, 1996, an oil pipeline operator in Fork Shoals, South Carolina acted on erroneous data that conflicted with the true state of the pipeline system [5]. To relieve pressure in the pipeline, the operator sent a remote signal to start a pump. Although the operator's console revealed that the pump had started, it was a faulty indication and the pump had not been activated. As the pressure readings continued to increase, the operator was confused by the anomaly and took actions that exacerbated the problem. The pipeline ultimately ruptured, spilling 957,600 gallons of oil into a nearby river and surrounding areas, and causing more than 20 million dollars in damage.

Industrial control systems monitor and control infrastructure assets that are vital to society – the electric power grid, oil and gas pipelines, transportation

systems and water treatment and supply facilities. Attacks that impact the operations of these critical assets can have devastating consequences. The complexity and interconnectivity of control systems have introduced vulnerabilities and attack surfaces that previously did not exist, resulting in a significant increase in security incidents during the past few years [6, 7]. Indeed, researchers have demonstrated that a number of critical infrastructure systems have been exposed to malicious process manipulation [1, 8].

Industrial control devices inherently trust system inputs for proper operation [4]. Few, if any, advanced decision support systems are available to assist operators in identifying anomalous data and determining the best course of action in the presence of conflicting information about process systems. As a result, accidental or malicious manipulations of system parameters can cascade to produce incorrect functionality and possibly induce system failures.

The Byzantine Generals Problem (BGP) [2] is a classic problem in distributed computing that seeks to determine the appropriate course of action when there is no consensus among the actors. Indeed, this problem is relevant to industrial control systems where operators often have to make important process control and management decisions in the presence of bad data. This paper considers a formulation of the Byzantine Generals Problem in the context of industrial control systems. The goal is to draw inferences from the physical state of a system to help determine integrity compromises.

2. Byzantine Generals Problem

The Byzantine Generals Problem was originally introduced as an abstract problem for understanding the reliability of computer systems and failures stemming from conflicting information [2]. The problem is described in the context of malicious actors who can modify messages to create discontinuity and conflict. In the classical formulation of the problem, Byzantine generals communicate with each another by messenger and must decide on a common course of action: attack or retreat. Messages can be manipulated by senders or while they are in transit from senders to receivers. Each receiver must gather and compare messages from all the neighboring generals before making a final decision to attack or retreat.

The original work by Lamport, *et al.* [2] evaluated solutions for resolving conflicting data. Each solution assumes different requirements, features and constraints when evaluating the overall reliability of a system. In the Byzantine Generals Problem, traditional oral messages sent between the generals correspond to the messages sent between computer systems. A common plan of action guarantees that a small number of "traitors" cannot negatively impact the system by enforcing a bad plan.

Valid solutions require more than two-thirds of the generals to be loyal. A valid solution enables generals to reach consensus or an agreed upon decision that cannot be negatively influenced by a limited number of traitors. Thus, when there are only three generals, no solution exists in the presence of even a single traitor. Lamport, *et al.* [2] proved that consensus can be reached when

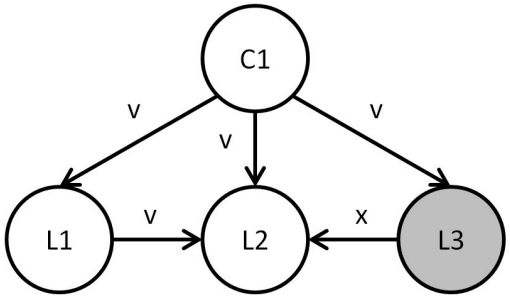

Figure 1. Byzantine Generals Problem.

there are at least $3m + 1$ generals in the presence of at most m traitors. More generally, with $3m + 1$ total nodes, at most m nodes can suffer from Byzantine faults. That is, for $m = 1$, only one of the four nodes can be malicious for the solution to be valid.

Figure 1 shows a traditional Byzantine Generals Problem scenario where the commander $C1$ sends a consistent value (message v) to three lieutenants, $L1$, $L2$ and $L3$, where $L3$ is a traitor. $L2$ receives conflicting data from the commander and the other two lieutenants, $C1$, $L1$ and $L3$, and evaluates the values provided by $C1$, $L1$ and $L3$. Specifically, $L2$ identifies the inconsistency using a majority function that considers the three inputs (v, v, x). The inconsistent data source is identified and, in this case, $L3$ is identified as the traitor based on the set of three messages. Note that the majority function is the basis for conflict resolution in the Byzantine Generals Problem [2].

3. Control Systems and Byzantine Failures

An industrial control system is a hierarchical, distributed system with operators, controllers and sensors, often many miles apart. The control system enables an operator at a distant location to assess the current status of a process and to perform the appropriate control actions to manage the process. This activity can be automated or semi-automated and, depending on the situation, can require frequent, regular or immediate intervention. Effective communications with field devices are critical for the proper operation of an industrial control system.

This section demonstrates how the Byzantine Generals Problem applies to an industrial control environment. Instead of the generals passing messages, field devices in an industrial control system pass messages to a control layer. Each field device collects state values from sensors and transmits the values to the control layer. The decision authority in the control layer executes an algorithm that compares the inputs received from the field (device) layer. The algorithm identifies malicious nodes and presents the current state of the system to enable an operator to make an informed decision when conflicting data is

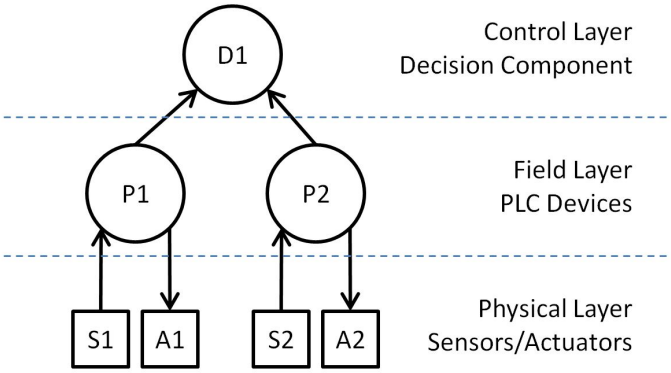

Figure 2. Industrial control system components.

received. This may require the operator to modify system parameters for better performance or to restore the system to a stable state if it has become unstable as a result of a data integrity compromise.

3.1 Basic Notions

In the Byzantine Generals Problem, a contributing node is defined as a node that produces a state value to pass directly to other available nodes. In the original formulation of the problem, all the nodes are contributing nodes, including the lead commander and the lieutenants. However, in an industrial control system, only field layer devices are contributing nodes.

A decision authority is defined as a node that evaluates inputs and decides on the state of the system. In the original Byzantine Generals Problem, all the nodes with the exception of the lead commander are decision authorities. However, an industrial control system has a single decision authority that resides in the control layer.

In an industrial control system, all the nodes are generally known and trusted. However, a node can be compromised and its data may be manipulated in an integrity attack.

Figure 2 shows the major components of an industrial control system corresponding to the Byzantine Generals Problem framework. The framework has the following types of nodes:

- **Decision Component Node:** This corresponds to a control layer node. A single decision component node, also known as a decision authority, is present. The decision component node receives state values from field device nodes and executes an algorithm to identify inconsistencies.
- **PLC Node:** This corresponds to a field layer node. Multiple PLC (programmable logic controller) nodes, also known as contributing nodes, are present. The PLC nodes send state values to the single decision component node. Note that only PLC devices are considered in this paper.

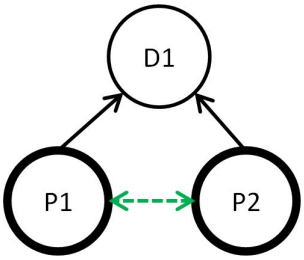

Figure 3. Industrial control system.

However, the approach is applicable to any type of industrial control device that reports status information to a central authority.

- **Sensor/Actuator Node:** This corresponds to a physical layer node. Multiple sensor (S_i) and actuator (A_i) nodes are present to monitor the process system and perform control actions on the process system, respectively. In an industrial control environment, the physical layer provides ground truth of the system state.

3.2 Industrial Control System Attributes

A Byzantine algorithm designed for an industrial control system must incorporate ground truth in order to correlate interdependencies between nodes. The ground truth is the actual state of a physical system. Given the state of one PLC, the decision component must infer the state of neighboring components. Inference is enabled by interdependent relationships between nodes. When neighboring components report state values, the decision component compares the reported state values with the anticipated state values.

Figure 3 shows a system with three nodes. Direct links exist from PLC nodes $P1$ and $P2$ to decision component node $D1$. Nodes $P1$ and $P2$ are contributing nodes.

To clarify the concepts, a simplified diagram of the original Byzantine Generals Problem is presented in Figure 4. The figure has three contributing nodes, $C1$, $L1$ and $L2$. The directional arrows in Figures 3 and 4 show the information flow between nodes. The solid lines in the two figures correspond to direct links along which messages are passed directly from one node to another; the messages may be manipulated by integrity attacks. The simplified industrial control system in Figure 3 uses a dotted line to represent an indirect link. The indirect link is not a physical link – it indirectly ties the two nodes together, enabling inferences to be made about the device state. Note that an indirect link cannot be compromised because it not a physical (actual) link.

To elaborate, the Byzantine Generals Problem in Figure 4 has direct links $C1 \rightarrow L1$, $C1 \rightarrow L2$, $L1 \rightarrow L2$ and $L2 \rightarrow L1$. The contributing nodes (bold circles) are $C1$, $L1$ and $L2$; the decision authorities are $L1$ and $L2$.

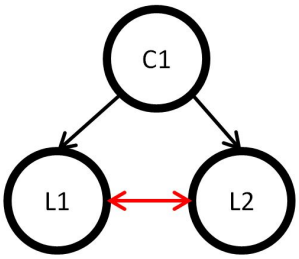

Figure 4. Byzantine Generals Problem.

The simplified industrial control system in Figure 3 has links $P1 \rightarrow D1$ and $P2 \rightarrow D1$. It has two contributing nodes, $P1$ and $P2$, which pass data to decision authority $D1$. Note that $D1$ is not a contributing node because it does not produce a state value that is sent to the other nodes.

The indirect link in Figure 3 is formed as a consequence of a direct physical relationship between the field devices. While direct links from contributing nodes can be compromised, an inferred link cannot be compromised; this reduces the attack surface. However, the interdependency captured by the inferred link enables the control layer to infer a change in state of a neighboring device. The interdependency arises from the ground truth of the system and the properties of dependent control system components.

As a practical example, consider a scenario where two sensors are attached to a bucket. One sensor detects the flow of water into the bucket while the other measures the weight of the bucket. If water does not leave the bucket as new water enters, the weight sensor should continually report an increase. The two sensors are, in fact, indirectly linked and their readings will always correlate if the system operates normally. However, if the system has an integrity error, the two sensors would not correlate; this inconsistency can lead to an unstable or undesired state. In the case of the bucket, the water will eventually overflow; in the Fork Shoals incident described above, the oil pipeline ruptured.

4. Algorithm

In order to detect integrity errors, the decision authority in an industrial control system executes an algorithm after it receives local state values from the contributing nodes. A total of $(l + m)$ state values exist, one from each of the l loyal contributing nodes and the m malicious contributing nodes. By using inferred data, the algorithm identifies the m malicious nodes, where $m \geq 1$ and the total number of nodes $n \geq 4$, as long as there are a majority of $l > m$ loyal nodes. To assist in identifying loyal and malicious nodes, the function CONSISTENT is invoked. This function evaluates the state values to identify the inconsistent nodes. After the inconsistent nodes are identified, the integrity of the entire system is evaluated by comparing the number of malicious nodes with the number of loyal nodes.

Each contributing node receives an input from a physical device that represents ground truth; however, the contributing nodes can be loyal or malicious. It is assumed that physical devices work properly and have no faults, and all the inconsistencies are due to the malicious nodes. It is also assumed that the decision node has complete information about the design of the physical system. Specifically, it can determine if the contributing nodes are reporting consistent values or inconsistent values.

The algorithm uses two primary functions. After all the state values are collected, the function CONSISTENT is executed for each pair of contributing nodes to determine consistency. After all the nodes are analyzed for consistency, the function MAJORITY is executed to determine if the majority of nodes are consistent or inconsistent.

The input to the CONSISTENT function is (s_i, s_{i+1}) where s_i and s_{i+1} are state values for PLCs P_i and P_{i+1}, respectively. There are two possible return values for the CONSISTENT function, True or False, which reflect whether the state values are consistent or inconsistent, respectively. If both the state values are consistent, then the state determination t_i is assigned the consistent value C; if the values are inconsistent, the inconsistent value I is assigned. Note that, if the values are consistent, then the nodes are either both loyal or both malicious. If the values are inconsistent, then one of the nodes is malicious.

The MAJORITY function evaluates the results generated by the CONSISTENT function. The function returns an overall state for the system. The system is consistent if, over all the t_i, the number of C values is greater that the number of I values; otherwise, the system is inconsistent.

4.1 Evaluation

The algorithm, shown in Figure 5, begins by acquiring local state values from all the $l + m$ contributing nodes. If there are at least three contributing nodes, the first value is labeled as consistent; otherwise, the algorithm terminates because of the lack of a sufficient number of loyal nodes required to evaluate system state. Next, pairs of state values are evaluated for consistency in sequential order. Nodes are labeled for consistency based on their relationship to previous evaluations. After all the nodes are evaluated for consistency, the majority operation is performed. If the majority of the nodes are consistent, then the first node is loyal. As a result, all the consistent nodes are labeled as loyal and all the inconsistent nodes are labeled as malicious. Alternatively, if the majority of the nodes are inconsistent, the first node is malicious. These results hold as long as there are more loyal nodes than malicious nodes (i.e., $l > m$).

Figure 6 shows a three-node system with $P1$ as a malicious node and flagged with an integrity error $t_1 = I$. In the example, decision authority $D1$ receives inputs from PLC field devices $P1$ and $P2$. $P1$ reports a local state value $s_1 = b$, where b is the value of a system parameter. Meanwhile, $P2$ reports a local state value $s_2 = d$. $D1$ can infer, based on the state value reported by $P1$, that the state value reported by $P2$ should correlate with the state

Step	Description
1	Decision component collects local state values For each PLC P_i, $1 \leq i \leq (l+m)$, let s_i be the local state value from P_i sent to the decision component d.
2	If there are at least three contributing nodes ($l+m \geq 3$), assume first node is consistent (t_1 = C); Otherwise algorithm terminates due to lack of sufficient nodes.
3	Compare all node values for consistency (relative to initial state) for i=1; $i<(l+m)$; i++ If *CONSISTENT*($s_i = s_{i+1}$) then ($t_{i+1} = t_i$); Else If t_i = I then t_{i+1} = C; Else t_{i+1} = I
4	Execute majority function on set of flagged states; label loyal and malicious nodes If *MAJORITY* ($t_1, t_2, \ldots, t_{(l+m)}$) = C, then first node is loyal; If *MAJORITY* ($t_1, t_2, \ldots, t_{(l+m)}$) = I, then first node is malicious;

Figure 5. Algorithm for detecting integrity errors.

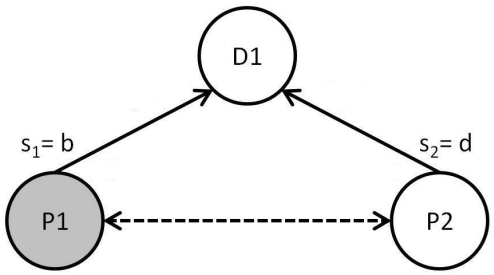

Figure 6. Impossibility of a solution for a system with less than four nodes.

c. Likewise, $D1$ can infer, based on the state value reported by $P2$, that the state value reported by $P1$ should be e. This discrepancy is caused by conflicting values from the two reporting field devices. An inconsistency is identified because CONSISTENT(s_1, s_2) = False. However, because of the small number of contributing nodes, $D1$ cannot determine the node that caused the integrity error.

The algorithm can accurately identify malicious nodes when there are a majority of loyal nodes ($l > m$). To demonstrate how the algorithm works, consider the basic case with $n = 4$. In this scenario, there are three contributing nodes and, as a result, $l > 1$ and $m < 2$ must be true for the algorithm to be successful.

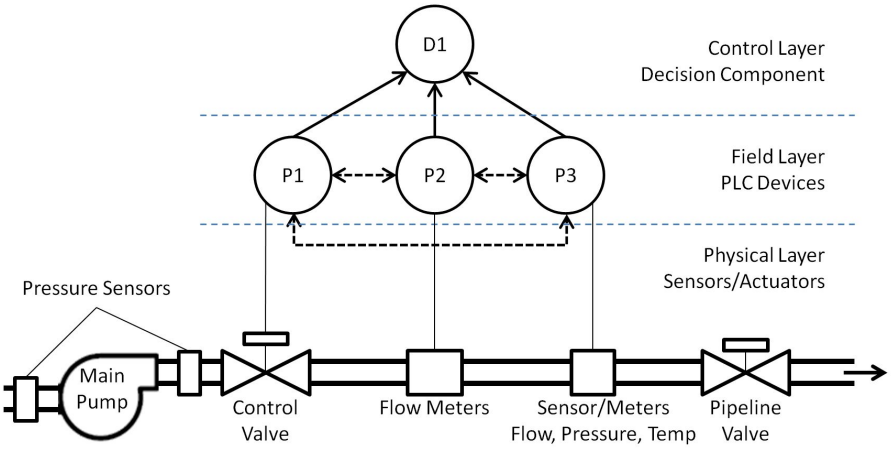

Figure 7. Oil pipeline representation.

Consider the case where $P3$ is flagged for an integrity problem. In the first step, $D1$ receives the state values: $s_1 = b$ from $P1$, $s_2 = c$ from $P2$ and $s_3 = f$ from $P3$. In the second step, since there are least three contributing nodes ($P1$, $P2$ and $P3$), the first node is labeled consistent ($t_1 = C$).

Next, the state values from $P1$ and $P2$ are compared for consistency. Specifically, CONSISTENT(s_1, s_2) = True. As a result, the integrity flag of $P2$ is set to the same value as $P1$ ($t_2 = C$). Next, CONSISTENT(s_2, s_3) = False. As a result, the integrity flag of $P3$ is set appropriately ($t_3 = I$).

Finally, the MAJORITY function is executed. Since MAJORITY(C, C, I) $= C$, the first node is loyal. As a result, all the nodes with $t_i = C$ are identified as loyal and all the nodes with $t_i = I$ are identified as malicious.

At this stage, using visual observations only, a control system operator may fail to identify the node with the integrity problem and could act on the invalid data, as in the case of the Fork Shoals pipeline rupture. Implementing this algorithm would have identified the faulty alert that led the pipeline operator to believe that the pump had started, when, in fact, it had not.

4.2 Application

The algorithm identifies integrity problems and a means for evaluating conflicting data. From a cyber security perspective, the integrity of field devices can be manipulated when components are networked to the Internet and targeted compromises or accidental manipulations occur. A field device can, thus, become compromised and report false data. If this occurs, the integrity of the data can be compromised. The following example highlights a scenario where devices provide inconsistent data, but the malicious device can be identified.

Figure 7 represents a notional oil pipeline with its associated connectivity and interdependencies. The physical layer components are several miles apart

Table 1. Sample oil pipeline data.

Pressure P1	**Valve P2**	**Flow P3**	**Consistent Yes/No**	**Malicious Node**
Low	Open	Yes	Yes	None
High	Open	Yes	No	$P1$
Low	Open	No	No	$P3$
High	Open	No	No	$P2$
Low	Closed	No	No	$P1$
High	Closed	No	Yes	None
Low	Closed	Yes	No	$P2$
High	Closed	Yes	No	$P3$

and the components are managed by multiple PLCs that report to a single decision component. The distribution of PLCs makes it difficult for an operator to manually or visually verify the current state of every device. Nonetheless, the operator must rely on the system for situational awareness prior to making decisions or taking actions. Previous examples have demonstrated the negative effects of conflicting data.

In the example, a field device $P1$ monitors pressure, a field device $P2$ monitors a control valve and a field device $P3$ monitors flow. A change in state at $P1$ changes the physical layer and the corresponding states of the subsequent field devices $P2$ and $P3$. This is an important property, which enables the decision authority to infer the state of the subsequent field devices. For simplicity, pressure can be High or Low, the valve position can be Open or Closed, and the flow sensor shows a flow state of Yes or No. Each field device reports either the accurate local state or the false local state.

Data reported by each field device is evaluated for consistency to allow the decision component $D1$ to make decisions. Integrity problems are present if the field devices $P1$, $P2$ and $P3$ report conflicting state values. Table 1 lists various combinations of sensor readings that represent consistent and inconsistent states in the notional example. The table values are used for evaluating the CONSISTENT function and enabling inference.

When the pressure at $P1$ is Low, the valve position at $P2$ should be Open and the flow rate at $P3$ should be Yes. When the valve position at $P2$ is Closed, the pressure at $P1$ is High and the flow rate at $P3$ is No. Inconsistent sequences of the reported state values are detected by the algorithm.

For example, consider the second row in Table 1. In the first step, the decision component receives state values from field devices $P1$, $P2$ and $P3$. $P1$ reports the pressure as High, $P2$ reports the valve as Open and $P3$ reports the flow rate as Yes. In the second step, $P1$ is labeled as consistent ($t_1 = C$). In the third step, values from $P1$ and $P2$ are compared for consistency. CONSISTENT(s_1, s_2) = False, meaning that High pressure at $P1$ does not infer an Open valve position at $P2$. Since $P1$ and $P2$ are inconsistent, $P2$ is labeled as inconsistent ($t_2 = I$). Next, values from $P1$ and $P3$ are compared

for consistency. CONSISTENT(s_1, s_3) = False, meaning a High pressure at $P1$ does not imply a flow rate of Yes at $P3$ and, therefore, $t_3 = I$. The comparison of state values from $P2$ and $P3$ reveals consistency. Since MAJORITY(C, I, I) = I, $P1$ is identified as malicious.

5. Conclusions

Human operators and automated decision components in industrial control environments often must make rapid decisions to react to system integrity errors. The application of the Byzantine Generals Problem to industrial control systems provides a formal mechanism for recognizing the presence of anomalous data and potentially identifying its sources. Using physical system properties, the resulting algorithm enables a decision authority to infer the system state and identify integrity compromises. A key constraint is that, when more than three field devices report the physical state of a system and when there are more trusted devices than compromised devices, it is possible to identify the specific devices that are compromised. The gas pipeline example demonstrates how the algorithm can identify and resolve conflicting data. As demonstrated, solutions to the Byzantine Generals Problem in the context of industrial control environments facilitate the resolution of inconsistent data that can result from cyber attacks against field devices and communications links.

References

[1] J. Finkle, "Irrational" hackers are growing U.S. security fear, *Reuters*, May 22, 2013.

[2] L. Lamport, R. Shostak and M. Pease, The Byzantine Generals Problem, *ACM Transactions on Programming Languages and Systems*, vol. 4(3), pp. 382–401, 1982.

[3] Y. Lindell, A. Lysyanskaya and T. Rabin, On the composition of authenticated Byzantine agreement, *Journal of the ACM*, vol. 56(6), pp. 881–917, 2006.

[4] T. Macaulay and B. Singer, *Cyber Security for Industrial Control Systems: SCADA, DCS, PLC, HMI and SIS*, CRC Press, Boca Raton, Florida, 2012.

[5] National Transportation Safety Board, Pipeline Accident Report, Pipeline Rupture and Release of Fuel Oil into the Reedy River at Fork Shoals, South Carolina, Report PB98-916502, NTSB/PAR-98-01, Washington, DC, 1996.

[6] F. Rashid, ICS-CERT: Response to cyber "incidents" against critical infrastructure jumped 52 percent in 2012, *Security Week*, January 10, 2013.

[7] U.S. Department of Homeland Security, ICS-CERT Incident Response Summary Report: 2009–2011, Washington, DC, 2012.

[8] Z. Zorz, Company's industrial heating system hacked via backdoor, *Help Net Security*, Kastav, Croatia, December 12, 2012.