Chapter 4

MODELING CONTROL SYSTEM FAILURES AND ATTACKS – THE WATERLOO CAMPAIGN TO OIL PIPELINES

Jonathan Butts, Mason Rice and Sujeet Shenoi

Abstract This paper presents a model for expressing control system failures and attacks on control protocols that involve the exchange of messages. Control failures and attacks are modeled using the notion of an attacker who can block and/or fabricate messages. These two attack mechanisms can cover a variety of scenarios ranging from control failures in the Waterloo Campaign to cyber attacks on oil pipelines. The model helps provide a comprehensive understanding of control system failures and attacks, which supports the development of strategies for attack as well as defense.

Keywords: Control systems, failure modeling, attack modeling

1. Introduction

Mankind's first conflicts were waged on land. As military technology advanced, battles were fought on sea and in the air. The earliest recorded naval battle occurred in 1210 BC when the Hittites led by Suppiluliumas II defeated a fleet from Cyprus. Aerial warfare was pioneered by the ancient Chinese who launched fire arrow attacks from "war kites" [1]; the first airplane bombing occurred in 1911 during the Libyan War between Italy and Turkey [1]; the first dogfights came soon after during World I. The first attack in space was a 1985 test that involved a U.S. F-15 shooting down a P78-1 communications satellite in a 345 mile orbit [15].

The 21st century brings a new dimension to the field of battle – cyberspace. Cyberspace, through its inextricable connection with the critical infrastructure, pervades all aspects of human endeavor – business, government and military operations, and societal functions. It is certain that future warfare will involve both cyberspace and the critical infrastructure.

Control is a vital component of any battle plan. A commander is responsible for controlling military operations. The commander must maintain constant situational awareness of the battlespace – allied forces, opposing forces, time and terrain. The commander uses various control techniques to maneuver forces to accomplish the mission within the battlespace parameters. Historians believe that Napoleon lost the Battle of Waterloo because of control failures made two days earlier during the Battles of Ligny and Quatre Bras [2, 11]. Sometimes, control failures are accidental; at other times, they are the result of the enemy compromising the control protocol. But however they occur, battles are won and lost because of control successes and failures.

This paper presents a model for expressing control system failures and attacks on control systems. Control system failures as well as attacks are modeled using the notion of an attacker who blocks and/or fabricates messages. In fact, these two types of attacks on control protocols can be used to express scenarios ranging from control failures during the Waterloo Campaign to cyber attacks on critical infrastructure assets such as oil pipelines.

The model, which is readily defined using graph theory, helps conceptualize attacks and failures in one control protocol and translate them to similar attacks and failures in another protocol. It also assists in targeting specific control protocol and system implementations. In particular, the model helps identify the information requirements, articulate possible outcomes and examine the feasibility of attacks based on the available information. The comprehensive understanding of attacks can facilitate risk analysis and risk management, the implementation of defensive postures and the design of robust control protocols.

2. The Waterloo Campaign

On June 16, 1815, two days before the pivotal Battle of Waterloo, Napoleon's troops were arrayed south of the road between the Belgian towns of Ligny and Quatre Bras (Figure 1). His 125,000 troops were divided into two commands [5]. Napoleon himself commanded the force on the east, just south of Ligny. Marshall Ney led the force to the west, a few miles south of Quatre Bras.

Napoleon's troops were opposed by two allied forces [5]. One force of 90,000 British, Dutch, Belgians and Germans commanded by the Duke of Wellington was positioned just north of Quatre Bras. The other force, a 115,000-man Prussian army led by Marshal Blucher, was positioned on the northern outskirts of Ligny.

The Battles of Ligny and Quatre Bras were fought that day. Napoleon won the Battle of Ligny; the Battle of Quatre Bras was a standoff [2]. Historians believe that the French forces could have crushed the opposition were it not for certain "control" failures [2, 11]. As a result, Wellington and Blucher were able to move north and reconstitute their forces. On June 18, 1815, the combined armies led by Wellington routed Napoleon's forces near the village of Waterloo.

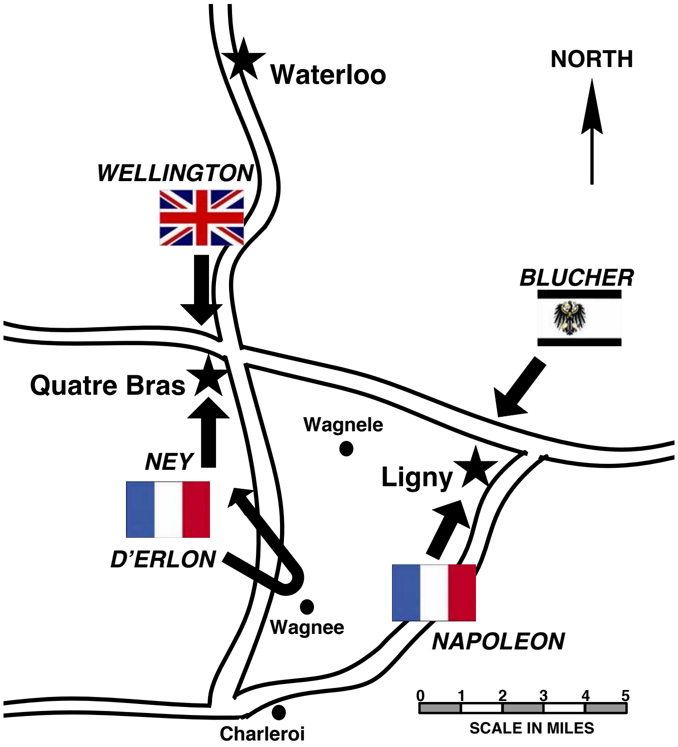

Figure 1. Battles of Ligny and Quatre Bras.

Before the Battles of Ligny and Quatre Bras, Napoleon had split his forces into two groups intending to drive a wedge between Wellington and Blucher (Figure 1). Napoleon wanted Ney to seize the crossroads at Quatre Bras while he destroyed Blucher's forces at Ligny [11]. The control failures occurred in the following chronological order:

- **Delay in Attacking Quatre Bras (Control Failure 1):** On the morning of June 16, 1815, Ney was in good position to take the strategic crossroads at Quatre Bras. Ney claimed that he did not receive the order to attack that morning. The Battle of Quatre Bras did not begin until 2 p.m. The delay gave Wellington time to place his troops in strong defensive positions.

- **Failure to Mobilize a Blocking Force (Control Failure 2):** Napoleon assumed that Ney would take the crossroads at Quatre Bras with ease. Shortly after 1 p.m., he ordered Ney to send a force towards Ligny to block the retreat of the Prussian forces involved in the Battle of Ligny. Ney did not receive the order. In any case, the Battle of Quatre Bras was still underway and Ney could not spare a blocking force.

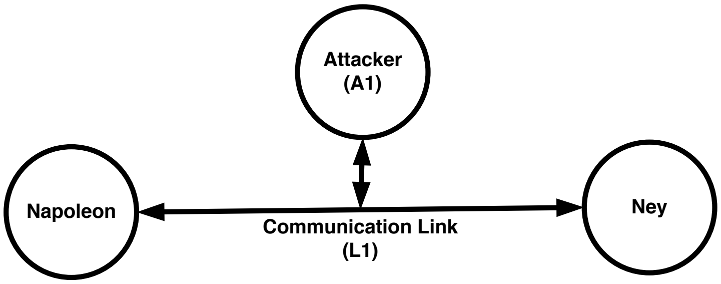

Figure 2. Communication pathway for Control Failures 1 and 2.

- **Failure to Engage a Reserve Force (Control Failure 3):** While the battle raged on at Quatre Bras, Napoleon's forces inflicted significant damage to the Prussians at Ligny. Sensing an opportunity for a devastating blow, Napoleon sent two messages: one to General d'Erlon ordering him to move his troops to Wagnele to attack the Prussian flank, and the other to Ney (d'Erlon's commander) informing him about the order. d'Erlon received Napoleon's message, but misread the message and marched towards the town of Wagnee instead of Wagnele.

  The message from Napoleon to Ney about moving d'Erlon's forces never arrived. When Ney learned that his reserve force under d'Erlon was moving away, he ordered it to turn back and support his forces at Quatre Bras. But it was too late and d'Erlon's reserve force neither joined the Battle of Ligny nor the Battle of Quatre Bras.

3. Modeling Napoleon's Control Failures

The control failures during the Battles of Ligny and Quatre Bras occurred as a result of delayed, lost and misinterpreted messages. The failures are attributed to the fog of war. However, we can formally model the failures using the notion of an attacker who blocks and/or fabricates messages. Indeed, these two types of attacks on messaging protocols can be used to express scenarios ranging from the control failures in the Waterloo Campaign to cyber attacks on critical infrastructure assets.

3.1 Modeling Control Failure 1

Figure 2 illustrates the communication pathway for Control Failure 1. Three nodes are involved: (i) Napoleon ($\in$ *ControlNodes*), (ii) Ney ($\in$ *EdgeNodes*) and (iii) A1 ($\in$ *AttackNodes*). Communications occur over L1 ($\in$ *Links*). In general, a link supports bidirectional message transfer, although each message is unidirectional from sender to receiver.

Control Failure 1 is consistent with a man-in-the-middle attack involving Attacker A1. In particular, Ney's delay in attacking Quatre Bras is modeled

System
Napoleon $\in$ $ControlNodes$
Ney $\in$ $EdgeNodes$
A1 $\in$ $AttackNodes$
L1 $\in$ $Links$
L1 = (A1: (Napoleon, Ney))
Request $\in$ $MsgTypes$

Capabilities
A1 = {¬, +}

Initial Node States
Ney := Ney^0
Napoleon := Ney^0

Possible Node States
Ney := Ney^0 | Ney^1
$\text{Ney}^0 \equiv$ Attack on Quatre Bras is FALSE
$\text{Ney}^1 \equiv$ Attack on Quatre Bras is TRUE
Napoleon := Ney^0 | Ney^1

Control Failure 1
1. Napoleon $\xrightarrow[A1\ \neg Request]{L1}$ Ney[Attack]
   Ney := Ney^0
   Napoleon := Ney^1
2. Napoleon $\xrightarrow[A1\ +Request]{L1}$ Ney[Attack]
   Ney := Ney^1
   Napoleon := Ney^1

Figure 3. Formal specification of Control Failure 1.

by A1 blocking Napoleon's message to Ney, then fabricating the same message and transmitting it to Ney some time later. This first step, a block (¬) of Napoleon's message, is represented as:

$$\text{Napoleon} \xrightarrow[A1\ \neg Request]{L1} \text{Ney[Attack]}.$$

Since Napoleon $\in$ *ControlNodes*, he can issue request messages (Request $\in$ *MsgTypes*) that are received and acted on by *EdgeNodes* like Ney. For a message to be valid, the nodes must have access to a common link (L1). In the man-in-the-middle attack, A1 $\in$ *AttackNodes* blocks (¬) the message sent from Napoleon to Ney along L1. At this point, Napoleon expects Ney to engage; however, Ney did not receive the message.

The second message transfer step in Control Failure 1 involves A1 fabricating a copy of Napoleon's original message (+) and sending it to Ney:

$$\text{Napoleon} \xrightarrow[A1\ +Request]{L1} \text{Ney[Attack]}.$$

Figure 3 shows the complete representation of Control Failure 1. The model includes the various nodes, links and message types. Attacker A1 has the capability to block (¬) and fabricate (+) messages. Note that L1 = (A1: (Napoleon, Ney)) expresses the fact that A1 has compromised Link L1 to perpetrate a man-in-the-middle attack between Napoleon and Ney. Ney's status is represented as one of two possible states: (i) Ney is not attacking Quatre Bras (Ney^0) or (ii) Ney is attacking Quatre Bras (Ney^1). Similarly, Napoleon maintains his perception of Ney's status (Ney^0 or Ney^1). The initial states for Ney and Napoleon are both Ney^0, i.e., Ney is not attacking Quatre Bras.

Figure 3 also shows the two-step message sequence for Control Failure 1. After Step 1 (message block), Napoleon assumes that Ney is attacking Quatre Bras (Napoleon := Ney^1) when, in fact, Ney is not attacking Quatre Bras (Ney := Ney^0). It is only after Step 2 (message fabrication) that Napoleon's and Ney's states match (Napoleon := Ney^1 and Ney := Ney^1).

System

Napoleon ∈ $ControlNodes$
Ney ∈ $EdgeNodes$
A1 ∈ $AttackNodes$
L1 ∈ $Links$
  L1 = (A1: (Napoleon, Ney))
Request ∈ $MsgTypes$

Capabilities

A1 = {¬}

Initial System State

Ney := Ney^0
Napoleon := Ney^0

Possible Node States

Ney := Ney^0 | Ney^1
  Ney^0 ≡ Blocking Force is FALSE
  Ney^1 ≡ Blocking Force is TRUE
Napoleon := Ney^0 | Ney^1

Control Failure 2

1. Napoleon $\xrightarrow[A1\ \neg Request]{L1}$ Ney[Attack]
  Ney := Ney^0
  Napoleon := Ney^1

Figure 4. Formal specification of Control Failure 2.

3.2 Modeling Control Failure 2

The failure of Napoleon to mobilize a blocking force against the retreating Prussians is modeled by Attacker A1 blocking Napoleon's order to Ney. The communication pathway is the same as that for Control Failure 1 (Figure 2).

Figure 4 shows the complete representation of Control Failure 2. The nodes, links and message types are the same as for Control Failure 1 (Figure 3). However, Attacker A1 only requires the capability to block (¬) messages. Ney's status is represented as one of two possible states: (i) Blocking force is not engaged (Ney^0) or (ii) Blocking force is engaged (Ney^1). Similarly, Napoleon maintains his perception of Ney's status (Ney^0 or Ney^1). The initial states for Ney and Napoleon are both Ney^0, i.e., Blocking force is not engaged.

The bottom half of Figure 4 shows the one-step message sequence corresponding to Control Failure 2. After Step 1 (message block), Napoleon assumes that Ney's blocking force is engaged (Napoleon := Ney^1) when, in fact, Ney's blocking force is not engaged (Ney := Ney^0).

3.3 Modeling Control Failure 3

Figure 5 shows the communication pathways involved in Control Failure 3. Note that Napoleon, Ney ∈ *ControlNodes*; d'Elron ∈ *EdgeNodes*; and A1, A2 ∈ *AttackNodes*. Communication occurs over three links: L1, L2, L3 ∈ *Links*. Links L1 and L2 are compromised by Attackers A1 and A2, respectively. Link L3 supports communication between Ney and d'Erlon, and is not compromised by an attacker.

Figure 6 shows the complete representation of Control Failure 3. Attacker A1 has the capability to block (¬) messages while Attacker A2 can block (¬) and fabricate (+) messages. d'Erlon's status is represented as one of three possible states: (i) d'Erlon is at Quatre Bras (d'Erlon^0), (ii) d'Erlon is at Wagnele (d'Erlon^1) or (iii) d'Erlon is at Wagnee (d'Erlon^2). Similarly, Napoleon and Ney each maintain their own perceptions of d'Erlon's status (d'Erlon^0, d'Erlon^1

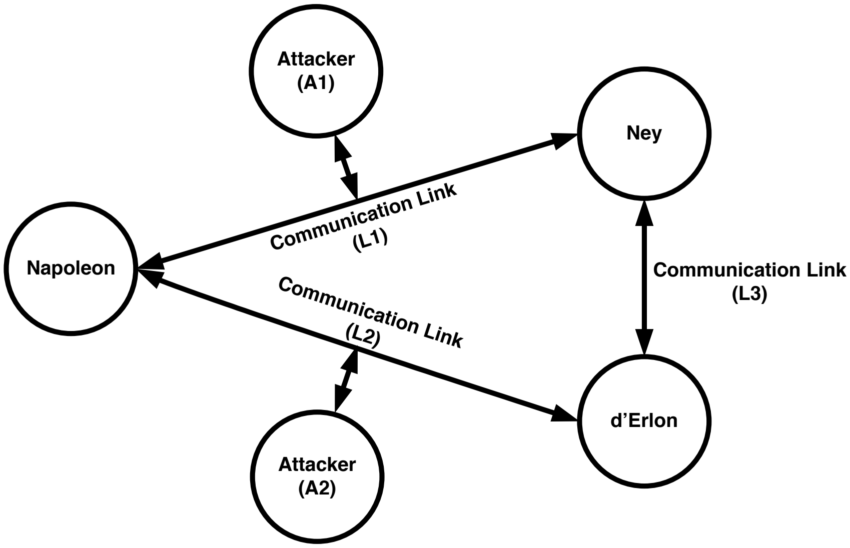

Figure 5. Communication pathways for Control Failure 3.

or d'Erlon^2). The initial states for d'Erlon, Ney and Napoleon are all d'Erlon^0, i.e., d'Erlon is staged at Quatre Bras.

The message sequences that result in Control Failure 3 are broken down into three processes. Processes 1 and 2 are independent and can occur in parallel; however, Process 3 must occur after Process 1.

The first step in Process 1 involves Attacker A2 blocking Napoleon's message to d'Erlon that orders his troops to Wagnele. After Step 1, Napoleon believes that d'Erlon is at Wagnele (Napoleon := d'Erlon^1); however, d'Erlon is still at Quatre Bras (d'Erlon := d'Erlon^0). Since Ney is not involved in the message exchange, his perception of d'Erlon's status is unchanged (Ney := d'Erlon^0). In Step 2, Attacker A2 sends a fabricated message to d'Erlon ordering him to move his troops to Wagnee (d'Erlon := d'Erlon^2). Napoleon still believes d'Erlon to be at Wagnele (Napoleon := d'Erlon^1) and Ney's perception of d'Erlon's status is unchanged (Ney := d'Erlon^0).

In Process 2, A1 blocks Napoleon's message to Ney that would have informed Ney of d'Erlon's movement. Process 2 may run concurrently with Process 1, implying that d'Erlon is either at Quatre Bras (d'Erlon := d'Erlon^0) or Wagnee (d'Erlon := d'Erlon^2) depending on the order of execution. The superscript ($*$) denotes that d'Erlon is in one of multiple possible states (d'Erlon := d'Erlon*). Note that the specific state is not relevant because d'Erlon would still be out of position.

Process 3 is conditional on Ney's observation that d'Erlon is moving away from Quatre Bras (Ney := d'Erlon^1 or Ney := d'Erlon^2). At this point, Ney sends a message to d'Erlon to reverse course and engage at Quatre Bras. Upon receiving this message, d'Erlon moves to Quatre Bras (d'Erlon := d'Erlon^0),

System

Napoleon, Ney $\in$ *ControlNodes*
d'Erlon $\in$ *EdgeNodes*
A1, A2 $\in$ *AttackNodes*
L1, L2, L3 $\in$ *Links*
  L1 = (A1: (Napoleon, Ney))
  L2 = (A2: (Napoleon, d'Erlon))
  L3 = (Ney, d'Erlon)
Request $\in$ *MsgTypes*

Capabilities

A1 = $\{\neg\}$
A2 = $\{\neg, +\}$

Initial System State

d'Erlon := d'Erlon^0
Ney := d'Erlon^0
Napoleon := d'Erlon^0

Possible Node States

d'Erlon :=
  d'Erlon^0 | d'Erlon^1 | d'Erlon^2
    d'Erlon^0 $\equiv$ Quatre Bras is TRUE
    d'Erlon^1 $\equiv$ Wagnele is TRUE
    d'Erlon^2 $\equiv$ Wagnee is TRUE
Ney :=
  d'Erlon^0 | d'Erlon^1 | d'Erlon^2
Napoleon :=
  d'Erlon^0 | d'Erlon^1 | d'Erlon^2

Process 1

1. Napoleon $\xrightarrow[A2\ \neg Request]{L2}$ d'Erlon[Wagnele]
  d'Erlon := d'Erlon^0
  Ney := d'Erlon^0
  Napoleon := d'Erlon^1
2. Napoleon $\xrightarrow[A2\ +Request]{L2}$ d'Erlon[Wagnee]
  d'Erlon := d'Erlon^2
  Ney := d'Erlon^0
  Napoleon := d'Erlon^1

Process 2

1. Napoleon $\xrightarrow[A1\ \neg Request]{L1}$ Ney[d'Erlon Wagnele]
  d'Erlon := d'Erlon*
  Ney := d'Erlon^0
  Napoleon := d'Erlon^1

Process 3

CONDITIONAL: IF (d'Erlon != d'Erlon^0)
1. Ney $\xrightarrow[Request]{L3}$ d'Erlon[Quatre Bras]
  d'Erlon := d'Erlon^0
  Ney = d'Erlon^0
  Napoleon = d'Erlon^1

Figure 6. Formal specification of Control Failure 3.

Ney believes that d'Erlon is moving to Quatre Bras (Ney := d'Erlon^0), but Napoleon believes that d'Erlon is at Wagnele (Napoleon := d'Erlon^1).

4. Formal Model

This section formalizes the approach used to express the control failures involved in the Battles of Ligny and Quatre Bras. The formal model is intended to express and reason about failures and attacks on SCADA systems used to control critical infrastructure assets.

Control protocols use messages to direct actions and provide feedback using a hierarchical, request-reply paradigm. Figure 7 shows a generic process diagram. *ControlNodes* send request messages to subordinate *EdgeNodes* or other control nodes (sub-control devices) to specify control actions and/or obtain data. *EdgeNodes* translate request messages into physical actions and/or physical actions into reply messages that are transmitted to their *ControlNodes*. In general, a message may involve a request-reply sequence, a request without a reply or an unsolicited reply. Request and reply messages are transmitted along bi-directional communication links that connect two or more nodes.

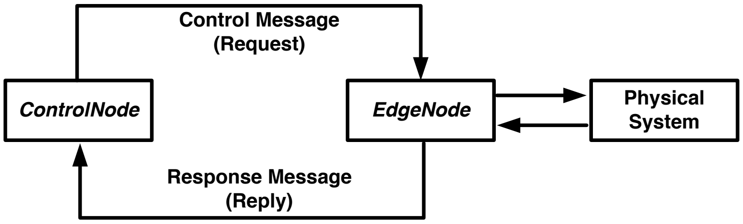

Figure 7. Generic process diagram.

A communication involves a sender transmitting a message to a receiver along a communication link:

$$MsgSource \underset{MsgTypes}{\overset{Link}{\longrightarrow}} MsgDest[Payload].$$

MsgSource and *MsgDest* are *ControlNodes* or *EdgeNodes* that communicate over the specified *Link*. *MsgType* is the type of message as defined by the control protocol (e.g., Request or Reply). *Payload* is the data contained in the message.

An attack on a control protocol occurs when an attacker (represented as an *AttackNode*) blocks legitimate messages or sends fabricated messages along a communication link. Formally, a message used in an attack is specified as:

$$MsgSource \underset{AttackNodes,Capabilities,MsgTypes}{\overset{Link}{\longrightarrow}} MsgDest[Payload].$$

MsgSource is the original sender or the spoofed sender of the attack message ($\in$ *ControlNodes* $\cup$ *EdgeNodes*). *MsgDest* is the intended target of the message ($\in$ *ControlNodes* $\cup$ *EdgeNodes*). The *AttackNode*, the perpetrator of the attack, has the *Capabilities* to block and/or fabricate messages along the *Link*. Note that message blocking and fabrication enable an attacker to launch a variety of messaging attacks. Message modification is implemented by blocking a legitimate message followed by sending a fabricated message. Message replay is implemented by sending a fabricated message with the same payload as an earlier message. Likewise, message delay is implemented by blocking a legitimate message followed by sending the original message some time later.

In general, an attacker has two attack avenues. The first is to compromise a *ControlNode* or *EdgeNode* and convert it into an *AttackNode*; this enables the attacker to block messages sent to the compromised node and to send fabricated messages from the compromised node. The second attack avenue is to compromise a link, which enables the attacker to perpetrate man-in-the-middle attacks. As shown in Figures 3, 4 and 6, the situation where Attacker A1 compromises Link L1 between Nodes N1 and N2 is expressed as L1 = (A1: (N1, N2)).

A node N has an initial state (N := N^p) and may change its state (N := N^q) upon receiving a message or as a result of a change in the physical state of the device expressed by the node (e.g., closed valve). Note that a superscript $(*)$ is used to denote that a node is in one of multiple possible states. A *ControlNode* also maintains information about the status of its subordinate *EdgeNodes*. For example, Z := (N1^p, N2^q) denotes that *ControlNode* Z perceives the states of subordinate nodes N1 and N2 to be N1^p and N2^q, respectively.

The formal model expresses temporal and causal properties using sequential steps, conditional statements and independent processes. A "process" is a sequence of messages that occur in a specific order; the process may be executed ψ number of times. A "conditional process" only executes when a Boolean condition holds.

5. Modeling an Attack on an Oil Pipeline

Oil pipelines often rely on SCADA systems to manage, direct and monitor large-scale, distributed operations. Similar to the Waterloo Campaign, control failures in these systems can result in devastating consequences.

This section describes and models a pipeline rupture incident that occurred at Fork Shoals, South Carolina on June 26, 1996. The rupture released 957,600 gallons of fuel oil and caused damage estimated at $20.5 million. According to the National Transportation Safety Board (NTSB) Pipeline Accident Report [8], the incident occurred as a result of failures in system components and improper operator actions. However, as we show in this section, the same results can be produced by targeted cyber attacks. For brevity, only the critical events that led to the pipeline rupture are discussed.

5.1 Pipeline Rupture Incident

The Fork Shoals pipeline transports fuel oil from Atlanta to Greensboro (North Carolina). Figure 8 shows the pipeline section of interest, which contains four pump stations (A–D), a delivery facility (F) with breakout tankage and a control center (Z) located in a central office north of the pipeline. Control Center Z houses operators that remotely monitor and control the pipeline using communication links (L1–L4). The remote pump stations (A–D) each have one RTU that controls actuators and reads pipeline sensors. Delivery Facility F is monitored by the control center, but the communication link is not shown because it is not pertinent to the analysis.

The pipeline rupture was due to two primary factors: (i) increase in pressure flow beyond the maximum allowable pipeline pressure, and (ii) failure of operator to realize and correct the conditions before structural failure occurred. The control failures that resulted in the pressure increase occurred in the following chronological order:

- **Increase in Pumping Capacity (Control Failure 1):** Pumping capacity is increased by starting additional pumps and/or turning on larger pumps and shutting down smaller ones. After a transfer to Delivery Facility F was completed, pumping capacity at downstream pumping facilities was sequentially increased to accommodate the additional fuel oil in the

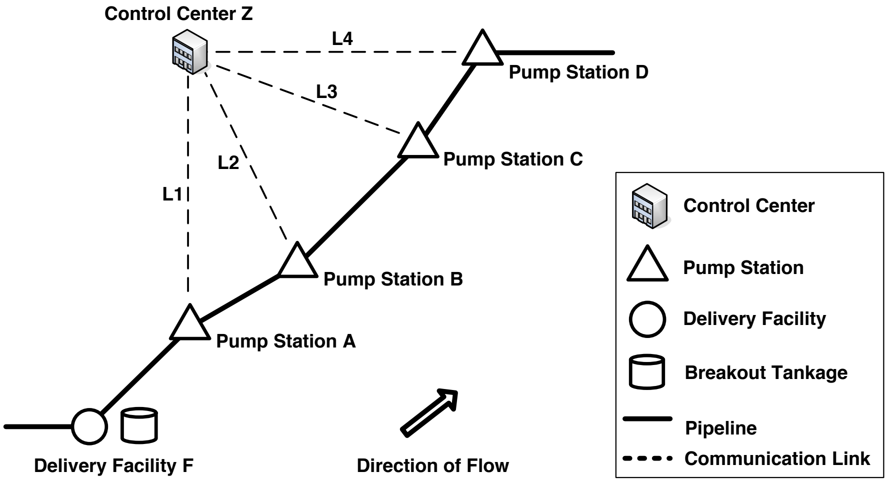

Figure 8. Pipeline layout.

pipeline. In particular, Pump Stations A and B each started a second pump. Also, Pump Station C started a larger pump and shut down a smaller one.

- **Failure to Start a Second Pump at Pump Station D (Control Failure 2):** The operator attempted to start a larger pump at Pump Station D to increase capacity. The operator noted that a green light appeared on the console to indicate that the pump had started, but, for some reason, the pump did not start.

- **Stoppage of the Active Pump at Pump Station D (Control Failure 3):** Believing that two pumps at Pump Station D were running, the operator stopped the smaller (and only operating) pump. Shutting down the only pump at Pump Station D created a pressure surge that traveled upstream to Pump Station C, causing its only operating pump to shut down due to high discharge pressure. The resulting second pressure surge caused the two pumps at Pump Station B to shut down. The continued high fuel oil flow rate caused the pipeline pressure to grow rapidly, resulting in a rupture between Pump Stations A and B.

Meanwhile, the pipeline operator at Control Center Z either ignored, misinterpreted or did not receive alarm notifications. The following control failure hindered the pipeline operator's situational awareness:

- **Failure to Receive and React to Alert Notifications (Control Failure 4):** A pressure alarm was triggered shortly after the pump at Pump Station B shut down; however, the operator took no action. Additionally, low suction pressure alarms were triggered intermittently for readings at Pump Station B, but the operator did not react because the alarms were behaving erratically and he assumed that the pressure readings were inaccurate. Moreover, the SCADA system did not report the failure of the pump to start at Pump Station D.

5.2 Cyber Attack Scenario

This section uses a cyber attack scenario to recreate the control system failures that led to the pipeline rupture. As shown in Figure 8, the system incorporates five nodes: Z (∈ *ControlNodes*) and A–D (∈ *EdgeNodes*), and four communication links L1–L4 (∈ *Links*).

The attacks described below involve compromising *ControlNode* Z and using it as the *AttackNode*. Essentially, the attacker has "root" access to Z, and can block (¬) and fabricate (+) messages.

Control Failure 1: This control failure occurred because of a series of messages that started and stopped various pumps along the pipeline. The attacker begins by fabricating (+) a message from Z to A to start the second pump (Pump #2):

$$\text{Z} \underset{Z\ +Request}{\overset{L1}{\longrightarrow}} \text{A[Start Pump \#2]}.$$

Pump Station A then generates an acknowledgment message to Z confirming that Pump #2 has started. In pipeline control protocols (e.g., Modbus), field (slave) devices typically send acknowledgements in response to requests from the control center (master); these acknowledgment messages must be blocked to mask the attack from the operator. Thus, the next step is to block (¬) the acknowledgment message from A to Z:

$$\text{A} \underset{Z\ \neg Reply}{\overset{L1}{\longrightarrow}} \text{Z[ACK]}.$$

Next, the attacker sends a fabricated message from Z to B to start Pump #2 and blocks the acknowledgement message:

$$\text{Z} \underset{Z\ +Request}{\overset{L2}{\longrightarrow}} \text{B[Start Pump \#2]},$$
$$\text{B} \underset{Z\ \neg Reply}{\overset{L2}{\longrightarrow}} \text{Z[ACK]}.$$

The attacker then starts a larger pump (Pump #3) at Pump Station C and blocks the acknowledgement message:

$$\text{Z} \underset{Z\ +Request}{\overset{L3}{\longrightarrow}} \text{C[Start Pump \#3]},$$
$$\text{C} \underset{Z\ \neg Reply}{\overset{L3}{\longrightarrow}} \text{Z[ACK]}.$$

Finally, the attacker stops the smaller pump (Pump #1) at Pump Station C and blocks the acknowledgement message:

$$\begin{aligned} \mathrm{Z} \underset{Z\ +Request}{\overset{L3}{\longrightarrow}} \mathrm{C[Stop\ Pump\ \#1]}, \\ \mathrm{C} \underset{Z\ \neg Reply}{\overset{L3}{\longrightarrow}} \mathrm{Z[ACK]}. \end{aligned}$$

Control Failure 2: This control failure occurred because the operator did not realize that the larger pump (Pump #3) at Pump Station D had not started despite directing it to start. The attacker implements this failure by ensuring that Pump #3 does not start and that the operator is unaware of this situation. Consequently, the attacker blocks a message from the Control Center Z to Pump Station D to start Pump #3 and fabricates an acknowledgement from D to Z that Pump #3 has started:

$$\begin{aligned} \mathrm{Z} \underset{Z\ \neg Request}{\overset{L4}{\longrightarrow}} \mathrm{D[Start\ Pump\ \#3]}, \\ \mathrm{D} \underset{Z\ +Request}{\overset{L4}{\longrightarrow}} \mathrm{Z[ACK]}. \end{aligned}$$

Control Failure 3: This control failure occurred because the operator stopped the only pump (Pump #1) at Pump Station D. The attacker implements this failure by fabricating a message to stop Pump #1 at Pump Station D and then blocking the acknowledgment message from D to Z:

$$\begin{aligned} \mathrm{Z} \underset{Z\ +Request}{\overset{L4}{\longrightarrow}} \mathrm{D[Stop\ Pump\ \#1]}, \\ \mathrm{D} \underset{Z\ \neg Reply}{\overset{L4}{\longrightarrow}} \mathrm{Z[ACK]}. \end{aligned}$$

Control Failure 4: This control failure occurred because the operator did not receive and react to alert notifications. Pressure fluctuations and pump shutdowns trigger alarms in SCADA systems. Alarms in typical SCADA systems are sent in the form of response messages from a field device to the master during normal polling cycles. The control failure is implemented by having the attacker block polling messages from Z to A–D and fabricate response messages to reflect normal operating conditions:

$$\begin{aligned} &\mathrm{Z} \underset{Z\ \neg Request}{\overset{L1}{\longrightarrow}} \mathrm{A[Poll]},\ \mathrm{A} \underset{Z\ +Reply}{\overset{L1}{\longrightarrow}} \mathrm{Z[ACK]}, \\ &\mathrm{Z} \underset{Z\ \neg Request}{\overset{L2}{\longrightarrow}} \mathrm{B[Poll]},\ \mathrm{B} \underset{Z\ +Reply}{\overset{L2}{\longrightarrow}} \mathrm{Z[ACK]}, \\ &\mathrm{Z} \underset{Z\ \neg Request}{\overset{L3}{\longrightarrow}} \mathrm{C[Poll]},\ \mathrm{C} \underset{Z\ +Reply}{\overset{L3}{\longrightarrow}} \mathrm{Z[ACK]}, \\ &\mathrm{Z} \underset{Z\ \neg Request}{\overset{L4}{\longrightarrow}} \mathrm{D[Poll]},\ \mathrm{D} \underset{Z\ +Reply}{\overset{L4}{\longrightarrow}} \mathrm{Z[ACK]}. \end{aligned}$$

5.3 Modeling the Cyber Attack Scenario

Figures 9 and 10 present the formal specification of the cyber attacks described above. Figure 9 specifies the nodes, attacker capabilities, and the possible and initial node states. The system has five nodes: Control Center Z,

System

Z ∈ *ControlNodes*
A, B, C, D ∈ *EdgeNodes*
Z ∈ *AttackNodes*
L1, L2, L3, L4 ∈ *Links*
L1 = (Z, A)
L2 = (Z, B)
L3 = (Z, C)
L4 = (Z, D)
Request, Reply ∈ *MsgTypes*

Capabilities

$Z = \{\neg, +\}$

Initial Node States

$A := A^0$
$B := B^0$
$C := C^0$
$D := D^0$
$Z := (A^0, B^0, C^0, D^0)$

Possible Node States

$A := A^0 \mid A^1 \mid A^2$
$A^0 \equiv (1, 0, 0, 0, 0)$
$A^1 \equiv (1, 1, 0, 0, 0)$
$A^2 \equiv (*, *, *, *, 1)$
$B := B^0 \mid B^1 \mid B^2$
$B^0 \equiv (1, 0, 0, 0, 0)$
$B^1 \equiv (1, 1, 0, 0, 0)$
$B^2 \equiv (*, *, *, *, 1)$
$C := C^0 \mid C^1 \mid C^2 \mid C^3$
$C^0 \equiv (1, 0, 0, 0, 0)$
$C^1 \equiv (1, 0, 1, 0, 0)$
$C^2 \equiv (0, 0, 1, 0, 0)$
$C^3 \equiv (*, *, *, *, 1)$
$D := D^0 \mid D^1 \mid D^2 \mid D^3$
$D^0 \equiv (1, 0, 0, 0, 0)$
$D^1 \equiv (0, 0, 1, 0, 0)$
$D^2 \equiv (0, 0, 0, *, *)$
$D^3 \equiv (*, 0, 1, 0, 0)$
$Z := (A^0 \mid A^1 \mid A^2,$
$B^0 \mid B^1 \mid B^2,$
$C^0 \mid C^1 \mid C^2 \mid C^3,$
$D^0 \mid D^1 \mid D^2 \mid D^3)$

Figure 9. Formal specification of the cyber attacks (nodes, capabilities and states).

which is both a *ControlNode* and an *AttackNode*, and Pump Stations A–D. The attacker has the capability to block (¬) and fabricate (+) messages.

Table 1 shows the possible states of Pump Stations A–D. The first three columns list the status of (small) Pump #1, (small) Pump #2 and (large) Pump #3. The fourth column shows whether or not the pressure reading is within acceptable limits. The fifth column indicates if the node is an alarm state. The status of a pump is represented as On (1) or Off (0). The pressure status is Acceptable (0) (i.e., within acceptable limits) or Not Acceptable (1). The alarm status is True (1) or False (0). Note that a "*" entry signifies that the specific binary value does not matter for that particular state.

The initial states for all four pump stations are identical:

$$A^0, B^0, C^0, D^0 \equiv (1, 0, 0, 0, 0).$$

This means that (small) Pump #1 is on, the other two pumps are off, the pressure readings are within acceptable limits and no alarms are triggered. The node states vary as different actions are performed along the pipeline.

The four control failures can be broken down into four processes: Processes 1, 2, 3 and 4. Process 1 (Figure 10) occurs only once, so $\psi = 1$. The second pump (Pump # 2) at Pump Stations A and B are activated. At Pump Station C, the small pump (Pump #1) is deactivated and the large pump (Pump #3) is activated unbeknownst to the operator.

Process 2 (Figure 10) corresponds to Control Failure 2. It is conditional on a message being sent to activate the large pump (Pump #3) at Pump Station

Table 1. Possible states of *EdgeNodes* (Pump Stations A–D).

State	**Pump #1**	**Pump #2**	**Pump #3**	**Pressure**	**Alarm**
A^0	On	Off	Off	Acceptable	F
A^1	On	On	Off	Acceptable	F
A^2	*	*	*	*	T
B^0	On	Off	Off	Acceptable	F
B^1	On	On	Off	Acceptable	F
B^2	*	*	*	*	T
C^0	On	Off	Off	Acceptable	F
C^1	On	Off	On	Acceptable	F
C^2	Off	Off	On	Acceptable	F
C^3	*	*	*	*	T
D^0	On	Off	Off	Acceptable	F
D^1	Off	Off	On	Acceptable	F
D^2	Off	Off	Off	*	*
D^3	*	Off	On	Acceptable	F

D. This condition always holds after the operator sends an activation message, so $\psi = \infty$.

Process 3 (Figure 10) corresponds to Control Failure 3. The steps deactivate Pump #1 at Pump Station D. Process 3, like Process 1, is executed once, so ψ = 1.

Process 4 (Figure 10) corresponds to Control Failure 4. Since polling is a continuous process, $\psi = \infty$. As discussed above, the attacker blocks polling messages from Z to A–D and fabricates response messages to reflect normal operating conditions despite the build-up of pressure that eventually causes the pipeline to rupture.

Note that a targeted cyber attack would simply turn off the pumps and mask the actions. However, the additional steps are incorporated in the example above to model the events described in the NTSB report.

6. Model Evaluation

This section discusses key applications of the model and its relationship to other work in the field.

6.1 Model Applications

The model was created specifically to express attacks on critical infrastructure assets. However, as demonstrated in the examples involving the Waterloo Campaign and pipeline rupture incident, the model can also be used to express failures in control protocols. In fact, the model is capable of expressing attacks and failures in diverse protocols that involve the exchange of messages.

Process 1
$\psi = 1$

1. Z $\xrightarrow[Z\ +Request]{L1}$ A[Start Pump #2]
A := A^1
Z := (A^0, B^0, C^0, D^0)

2. A $\xrightarrow[Z\ \neg Reply]{L1}$ Z[ACK]
A := A^1
Z := (A^0, B^0, C^0, D^0)

3. Z $\xrightarrow[Z\ +Request]{L2}$ B[Start Pump #2]
B := B^1
Z := (A^0, B^0, C^0, D^0)

4. B $\xrightarrow[Z\ \neg Reply]{L2}$ Z[ACK]
B := B^1
Z := (A^0, B^0, C^0, D^0)

5. Z $\xrightarrow[Z\ +Request]{L3}$ C[Start Pump #3]
C := C^1
Z := (A^0, B^0, C^0, D^0)

6. C $\xrightarrow[Z\ \neg Reply]{L3}$ Z[ACK]
C := C^1
Z := (A^0, B^0, C^0, D^0)

7. Z $\xrightarrow[Z\ +Request]{L3}$ C[Stop Pump #1]
C := C^2
Z := (A^0, B^0, C^0, D^0)

8. C $\xrightarrow[Z\ \neg Reply]{L3}$ Z[ACK]
C := C^2
Z := (A^0, B^0, C^0, D^0)

Process 2
$\psi = \infty$
CONDITIONAL:
IF (Z==(A^*, B^*, C^*, D^0)
or (A^*, B^*, C^*, D^2))

1. Z $\xrightarrow[Z\ \neg Request]{L4}$ D[Start Pump #3]
D := D^*
Z := (A^0, B^0, C^0, D^3)

2. D $\xrightarrow[Z\ +Reply]{L4}$ Z[ACK]
D := D^*
Z := (A^0, B^0, C^0, D^3)

Process 3
$\psi = 1$

1. Z $\xrightarrow[Z\ +Request]{L4}$ D[Stop Pump #1]
D := D^2
Z := (A^0, B^0, C^0, D^0)

2. D $\xrightarrow[Z\ \neg Reply]{L4}$ Z[ACK]
D := D^2
Z := (A^0, B^0, C^0, D^0)

Process 4
$\psi = \infty$

1. Z $\xrightarrow[Z\ \neg Request]{L1}$ A[Poll]
A := A^*
Z := (A^0, B^0, C^0, D^0)

2. A $\xrightarrow[Z+Reply]{L1}$ Z[ACK]
A := A^*
Z := (A^0, B^0, C^0, D^0)

3. Z $\xrightarrow[Z\ \neg Request]{L2}$ B[Poll]
B := B^*
Z := (A^0, B^0, C^0, D^0)

4. B $\xrightarrow[Z+Reply]{L2}$ Z[ACK]
B := B^*
Z := (A^0, B^0, C^0, D^0)

5. Z $\xrightarrow[Z\ \neg Request]{L3}$ C[Poll]
C := C^*
Z := (A^0, B^0, C^0, D^0)

6. C $\xrightarrow[Z+Reply]{L3}$ Z[ACK]
C := C^*
Z := (A^0, B^0, C^0, D^0)

7. Z $\xrightarrow[Z\ \neg Request]{L4}$ D[Poll]
D := D^*
Z := (A^0, B^0, C^0, D^0)

8. D $\xrightarrow[Z+Reply]{L4}$ Z[ACK]
D := D^*
Z := (A^0, B^0, C^0, D^0)

Figure 10. Formal specification of Control Failures 1, 2, 3 and 4.

The model provides a powerful mechanism for conceptualizing attacks and failures in one control protocol and translating them to similar attacks and failures in another protocol. For example, the sequence of attacks involved in rupturing a pipeline is very similar to the attack sequences that turn off a section of the electric power grid or shut down telephone service. The nodes (pumping stations, generators and service switching/transfer points) and protocols (Modbus, DNP3 and SS7) for pipelines, power grid and telecommunications infrastructures are different, but the attack strategies are practically identical and merely involve different messages with different payloads.

Techniques for masking attacks are just as similar across critical infrastructures. For example, the attack on the polling mechanism in the oil pipeline example is applicable to numerous protocols in the oil and gas sector (e.g., Modbus and Fisher ROC) and to protocols in other sectors such as the electric power grid (DNP3) and manufacturing (Profibus). The underlying strategy is to block polling messages and fabricate normal responses to mask alert conditions. Some critical infrastructure protocols (e.g., DNP3) support additional communication modes (e.g., unsolicited replies), but these modes are readily accommodated by the model.

In addition to conceptualizing common attacks and attack strategies for different protocols, the formal model assists in targeting specific protocol and system implementations. Developing attacks requires detailed information about the control system as well as the underlying cyber-physical systems. The model helps identify the information requirements, articulate possible outcomes and examine the feasibility of attacks based on the available information.

The formal model provides a framework for infrastructure asset owners and operators to evaluate system implementations and configurations for possible weaknesses. Moreover, the comprehensive understanding of attacks supports risk analysis and risk management, and the implementation of defensive postures. In particular, common vulnerabilities derived via attack analysis can be grouped into general security dimensions to aid the development and deployment of mitigation strategies. The formal model also enables researchers to analyze existing control protocols and to evaluate the implications of design decisions in new protocols.

6.2 Comparison With Other Work

Numerous models have been developed for expressing and reasoning about the security properties of computer systems and protocols. This section compares the proposed model with some of the established approaches for modeling attacks on computer and control systems.

The majority of attack models for computer systems are founded on the notion of an attack tree [12]. The root node of an attack tree denotes the goal of the attacker. The steps for completing the attack are decomposed using parent-child relations such that a path from a leaf node to the root node expresses one instance of the attack. The possible attacks correspond to the different branches of the attack tree. Attack trees have been shown to capture a variety of computer system attack scenarios [7, 17].

Attack tree models are attractive because of their simplicity and ease of analysis. However, certain ambiguities and the lack of expressiveness of attack tree models hinder quantitative reasoning and comparison. Precise analysis is difficult because an attack tree node represents both the current system state and the specific attack action [19]. Also, attack trees are susceptible to state explosion, they do not represent the temporal aspects of dependent attacks, and lack the ability to generalize beyond the modeled scenario [16].

An alternative model [16] views an attack as a series of capabilities instead of a sequence of events. It provides constructs for defining intrusion signatures and automating the discovery of attacks. Other researchers have proposed taxonomies for classifying attacks [9] and ontology-based models [18]. These models describe complex scenarios involving multiple attacks; however, they lack formalisms for analyzing causality and sequential events. Additionally, in these models, attacks focus on peer-to-peer communications and do not arise from a holistic view of the system. Attack models for control systems require the ability to express hierarchical communications, logical sequences of events and system-wide situational awareness.

Models based on finite state machines allow the formal representation of discrete-event, dynamic systems. Finite state machines facilitate the logical analysis of deterministic and non-deterministic attributes. Finite state machine models based on Petri nets have been used to express common computer attacks [6, 19]. Petri nets use graphs and set theory to model concurrency, synchronization, resource allocation and randomness. Our formal model specifies attack sequences and events that may be expressed and analyzed using Petri nets or other graph theoretical constructs.

Models for failures in control systems have been developed primarily to analyze reliability, resilience, functionality and risk. The U.S. Department of Energy [10], National Institute of Standards and Technology [14] and other entities [13] have developed models for predicting, reacting to and understanding failures in control systems. Several failure analysis models (e.g., [4]) demonstrate the conditions that can lead to failures in control systems. Our model does not directly focus on failure analysis. However, a failure analysis model can be used to determine the conditions under which physical damage can occur. The conditions can then be analyzed using our model to determine how a control protocol may be attacked to cause physical damage.

Little, if any, research has specifically focused on modeling attacks on control system protocols. Current work has tended to focus on modeling intrusion and anomaly signatures, identifying attributes for system resilience and evaluating system vulnerabilities for risk analysis. Cheung, *et al.* [3] describe a language for modeling attacks on process control systems. Because the systems are relatively static, models can be constructed to characterize the expected system behavior. Traffic that does not conform to normal traffic patterns is identified as a potential attack. The concept focuses on a purely defensive posture that assumes attackers will not use legitimate traffic in an attack. Our model is attack-centered in that it allows legitimate (as well as non-standard) messages in attack scenarios. Moreover, our model expresses temporal aspects and causal effects while lending itself to formal analysis.

7. Conclusions

The model presented in this paper can express control system failures and attacks on control protocols that involve the exchange of messages. The model helps provide a comprehensive understanding of control system failures and attacks, which supports the development and analysis of attack and defense strategies.

Our future research will concentrate on developing a graph-theoretic model augmented with temporal and belief attributes. The extended model will permit the specification of common modes of attack (e.g., control node compromise, edge node compromise, malicious control of edge nodes and polling mechanism manipulation). The model will also facilitate the development of attack metrics and will support formal reasoning about attacks and defensive strategies.

References

[1] J. Buckley, *Air Power in the Age of Total War*, Indiana University Press, Bloomington, Indiana, 1999.

[2] D. Chandler, *Waterloo – The Hundred Days*, Macmillan, New York, 1980.

[3] S. Cheung, U. Lindqvist and M. Fong, Modeling multistep cyber attacks for scenario recognition, *Proceedings of the Third DARPA Information Survivability Conference and Exposition*, pp. 284–292, 2003.

[4] L. Decker, A risk assessment model for pipeline facility operations, *Pipeline and Gas Journal*, vol. 236(3), pp. 38–44, 2009.

[5] HowStuffWorks.com, Battle of Waterloo, Atlanta, Georgia (history.how stuffworks.com/european-history/battle-of-waterloo.htm), 2008.

[6] J. McDermott, Attack net penetration testing, *Proceedings of the New Security Paradigms Workshop*, pp. 15–21, 2000.

[7] A. Moore, R. Ellison and R. Linger, Attack Modeling for Information Security and Survivability, Technical Note CMU/SEI-2001-TN-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2001.

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

[9] P. Neumann and D. Parker, A summary of computer misuse techniques, *Proceedings of the Twelfth National Computer Security Conference*, pp. 396–407, 1989.

[10] North American Electric Reliability Corporation, Reliability Functional Model, Function Definitions and Functional Entities, Version 5, Princeton, New Jersey (www.nerc.com/fileUploads/File/Standards/Functional_Model_V5_Clean_2009Sept24.pdf), 2009.

[11] A. Roberts, *Waterloo – June 18, 1815: The Battle for Modern Europe*, HarperCollins, New York, 2005.

[12] B. Schneier, Attack trees, *Dr. Dobb's Journal*, vol. 24(12), pp. 21–29, 1999.

[13] J. Stamp, M. Berg and M. Baca, Reference Model for Control and Automation Systems in Electrical Power, Version 1.2, Sandia National Laboratories, Albuquerque, New Mexico (www.oe.energy.gov/DocumentsandMedia /Reference_Model_for_Control_and_Auto_Systems_in_Elec_Ind.pdf), 2005.

[14] K. Stouffer, J. Falco and K. Scarfone, Guide to Industrial Control Systems Security, Final Public Draft, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2008.

[15] A. Tan, G. Badhwar, F. Allahdadi and D. Medina, Analysis of Solwind fragmentation event using theory and computations, *Journal of Spacecraft and Rockets*, vol. 33(1), pp. 79–85, 1996.

[16] S. Templeton and K. Levitt, A requires/provides model for computer attacks *Proceedings of the New Security Paradigms Workshop*, pp. 31–38, 2000.

[17] C. Ten, C. Liu and M. Govindarasu, Vulnerability assessment of cybersecurity for SCADA systems using attack trees, *Proceedings of the IEEE Power Engineering Society General Meeting*, pp. 1–8, 2007.

[18] J. Undercoffer, J. Pinkston, A. Joshi and T. Finin, A target-centric ontology for intrusion detection, *Proceedings of the IJCAI Workshop on Ontologies and Distributed Systems*, pp. 47–58, 2004.

[19] R. Wu, W. Li and H. Huang, An attack modeling based on hierarchical colored Petri nets, *Proceedings of the Second International Conference on Computer and Electrical Engineering*, pp. 918–921, 2008.