5-1-2016

TEDDI: Tamper Event Detection on Distributed Cyber-Physical Systems

Jason O. Reeves
Dartmouth College

TEDDI: Tamper Event Detection on Distributed Cyber-Physical Systems

Dartmouth Computer Science Technical Report TR2016-804

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Jason Reeves

DARTMOUTH COLLEGE

Hanover, New Hampshire

May 2016

Examining Committee:

_____

(chair) Sean W. Smith, Ph.D.

_____

Sergey Bratus, Ph.D.

_____

David F. Kotz, Ph.D.

_____

Zbigniew T. Kalbarczyk, Ph.D.

_____

Ryan Bradetich, Ph.D.

_____

F. Jon Kull, Ph.D.
Dean of Graduate Studies

Dedicated to "Mammie"

Sylvia Lillian Rathburn

1937 – 2016

Abstract

Edge devices, or embedded devices installed along the periphery of a power grid SCADA network, pose a significant threat to the grid, as they give attackers a convenient entry point to access and cause damage to other essential equipment in substations and control centers. Grid defenders would like to protect these edge devices from being accessed and tampered with, but they are hindered by *the grid defender's dilemma*; more specifically, the range and nature of tamper events faced by the grid (particularly distributed events), the prioritization of grid availability, the high costs of improper responses, and the resource constraints of both grid networks and the defenders that run them makes prior work in the tamper and intrusion protection fields infeasible to apply.

In this thesis, we give a detailed description of the grid defender's dilemma, and introduce TEDDI (Tamper Event Detection on Distributed Infrastructure), a distributed, sensor-based tamper protection system built to solve this dilemma. TEDDI's distributed architecture and use of a factor graph fusion algorithm gives grid defenders the power to detect and differentiate between tamper events, and also gives defenders the flexibility to tailor specific responses for each event. We also propose the TEDDI Generation Tool, which allows us to capture the defender's intuition about tamper events, and assists defenders in constructing a custom TEDDI system for their network.

To evaluate TEDDI, we collected and constructed twelve different tamper scenarios, and show how TEDDI can detect all of these events and solve the grid defender's dilemma. In our experiments, TEDDI demonstrated an event detection accuracy level of over 99% at both the information and decision point levels, and could process a 99-node factor graph in under 233 μs. We also analyzed the time and resources needed to use TEDDI, and show how it requires less up-front configuration effort than current tamper protection solutions.

Acknowledgements

After eleven years at Dartmouth, the list of people that deserve recognition for helping me reach this point is longer than my actual dissertation! I will never be able to thank them all for their support and guidance over the years, but I will do my best to try.

First, I would like to thank my graduate advisors, Sean Smith and Sergey Bratus. Sergey introduced me to the hacking community, and championed the Autoscopy Jr. project all the way to its inclusion in the SEL product line, while Sean helped me navigate navigate the field of tamper detection, and served as a pillar a support and source of encouragement throughout the design and development of TEDDI. The wisdom you both have shared with me, both about academia and life in general, has been invaluable, and I someday hope to be half the leader and mentor that you both are.

I would also like to thank the other members of my thesis committee, Zbigniew Kalbarczyk, Ryan Bradetich, and Dave Kotz: Your insight and encouragement played a major role in the success of TEDDI. Zbigniew introduced me to factor graphs as a powerful alternative to Bayesian networks, Ryan introduced us to this problem space and gave us the idea that would eventually grow into TEDDI, and Dave's advice and feedback pushed me to become a better researcher and make TEDDI a better project. This thesis would not have been possible without you all, and I am grateful for your support.

I would like to thank Bill Nisen, Tom Candon, Karen Page, and all the rest of my colleagues at the Institute for Security, Technology, and Society for always being there when I needed someone to bounce ideas off of, talk me back off the ledge when things seemed bleak, or listen to my rants about the deficiencies of the Orioles' pitching staff. I am a better student and person—and TEDDI is a better thesis—because of all of you. The ISTS is a valuable resource, and I only wish that more people realized it.

To Bx, Shrirang, Aarathi, Ray, Vijay, Tim, Travis, Stefan, Michael, Prashant, Vineetha, Pete, Gabe, Scout, Max, Joe, John, Rouslan, Ivan, Tucker, David x 2, Alex, Ryan, Ricky,

and at least twenty other people I've forgotten who have passed through the Trust and Kotz Labs during my tenure: Your presence and spirit went a long way towards making the pain of graduate school a lot more bearable. Working, talking, and just getting to hang out with you all has been an honor and a privilege. I owe a special thank you to Chris Frangieh for helping me design and build the TEDDI Generation Tool (and tolerating my bumbling attempts at being a project manager), and to Shrirang Mare for proofreading this document and making it readable—trust me, if you're reading this, you owe Shrirang a thank-you too.

To my homeboys Alex and Chris Tausanovitch: Twenty years ago we were nerdy outcasts playing Super Smash Brothers in your basement. Now we've got two PhDs and a law degree between us. How did this happen!?

Finally, the biggest thank you of all goes to my family: Bill, Debra, Joel, Erika, Otis, Sylvia, and Maggie. Thank you for supporting, encouraging, and putting up with me for over thirty years. I could not have done it without you.

This material is based upon work supported by the Department of Energy under Award Numbers DE-OE0000097 and DE-OE0000780.

(P.S. For the record, Kyle was no help at all.)

Disclaimer

Contents

List of Acronyms

ART	Attack Response Tree
BDD	Binary Decision Diagram
BN	Bayesian Network
CFI	Control-Flow Integrity
CIP	Critical Infrastructure Protection
CPTL	Cyber-Physical Topology Language
DRAM	Dynamic Random Access Memory
DSL	Domain-Specific Language
EM	Energy Management
FGDSL	Factor-Graph Domain-Specific Language
FIPS	Federal Information Processing Standards
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
JSON	JavaScript Object Notation
kV	Kilovolt
MAC	Message Authentication Code
MRF	Markov Random Field
NIPS	Network Intrusion Protection System

NIST National Institute of Standards and Technology

PAC Probabilistic Alert Correlation

PDP Policy Decision Point

PEP Policy Enforcement Point

PIP Policy Information Point

PQS Process Query Systems

RRE Response and Recovery Engine

TDP Tamper Decision Point

TEDDI Tamper Event Detection on Distributed Infrastructure

TEP Tamper Enforcement Point

TIP Tamper Information Point

SCADA Supervisory Control and Data Acquisition

SCPSE Security-Oriented Cyber-Physical State Estimation

SECaaS Security-as-a-Service

SPLP Simple Plant Location Problem

XACML eXtensible Access Control Markup Language

XML eXtensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we introduce TEDDI (Tamper Event Detection on Distributed Infrastructure), a novel, distributed, sensor-based tamper protection architecture to address security issues arising from the installation of *edge devices*, or networked smart devices on the periphery of utilities' networks. We show how these edge devices lead to a problem we define as *the grid defender's dilemma*; that is, these edge devices provide an easy way for an attacker to cause massive damage to the grid, and while power grid defenders[1] would like to use state-of-the-art protection solutions to prevent this damage, the unique goals and constraints of the power grid prevent defenders from doing so. We outline the various components of TEDDI and how they interact with one another, and describe how the system uses factor graphs [42] to make decisions about tamper events that are currently occurring based on the available data. Finally, we evaluate the speed, accuracy, and resource requirements of TEDDI, and show that TEDDI can solve the above dilemma with comparable or better performance than prior work.

[1] We use the terms *user*, *operator*, and *defender* interchangeably in this thesis to describe the utility personnel who are responsible for the security and proper operation of the power grid (and thus are the users we target with TEDDI).

1.1 Edge Devices and the Power Grid

A power grid is a large, interconnected web of power lines and substations designed to transport electricity from the power plants that generate it to the homes and businesses that consume it [139]. While the U.S. grid began as a collection of isolated utilities distributing power within their local area, as the demand for power grew, these utilities began to connect to each other to share the costs of building larger generation plants while also increasing the reliability of their local grids (since a utility could now draw on reserves from other utilities to meet demand when necessary). The result is our current grid setup, as described by the U.S. Energy Information Administration:

> *"The interconnected [power grid] systems now include about 2,000 electric distribution utilities, more than 300,000 miles of transmission and distribution lines, millions of customers, and more than 7,200 power plants and generating facilities that each has at least 1 megawatt of generating capacity"* [139].

Because electricity cannot be stored effectively in large quantities, the power grid is constructed as a *real-time* system; that is, power must be generated on-demand the moment it is needed [139]. As a real-time system, the grid requires constant monitoring and timely interventions to operate properly. Given the speed and destructive power of electricity, these interventions must happen quickly, often within a small time window in which a human cannot react. The need for such timely interventions has led the industry to automate many routine tasks based on the state of the grid, using devices such as generator governors and protective relays [143].

Recently, utilities have introduced a number of "smart" grid technologies into the grid, with the aim of improving the grid's reliability and efficiency by cutting down on power losses, reducing maintenance times, and encouraging consumers to save energy [139]. As part of this push towards a smarter grid, utilities have installed a number of *edge devices*

on their SCADA[2] networks. networks. Edge devices are resource-constrained embedded devices that live on the periphery of a network, e.g. at a consumer's home or on a telephone pole. An example of an edge device is a recloser control [113], which is used to configure how a utility's reclosers[3] behave when a fault is detected in the power lines, and is often mounted inside boxes on utility poles in the field.

Edge devices present a major security challenge for the power grid for three reasons:

1. **Distributed:** These devices are distributed all across a utility's service area and may appear in almost any environment, from remote rural areas (where these boxes are under very little supervision) to highly-populated urban areas (where they are easily accessible to a large number of people).

2. **Minimal Physical Security:** These devices often have little in the way of physical security. Generally, they are either directly exposed to the environment, or are locked within an easily-accessible cabinet.

3. **Network Access:** These devices have a direct connection to a utility's SCADA network. While this is intended to allow the device to communicate with specific parties, such as a data aggregator or a server at a utility's control center, it may also grant the device access to everything else on the network, such as other edge devices, other control centers, or perhaps other pieces of the control infrastructure.

In short, edge devices are enticing targets for malicious actors, who can use them to harm the grid on a number of levels:

At the Device Level: An attacker can access potentially sensitive data on the device (for example, cryptographic keys) or modify the outputs to mislead network operators

[2]SCADA stands for "Supervisory Control and Data Acquisition," and is generally used to describe the command-and-control networks used by critical industries like the power grid.

[3]A *recloser* is a grid protection device that is used to quickly restore service in the event of a transient fault, such as a short circuit [1]. When a transient fault is detected on a power line, a recloser can disconnect the line, wait for the fault to clear, and then reconnect the line automatically, which leads to shorter outages and fewer technician visits to field equipment.

(for example, report incorrect usage data to commit fraud).

At the Local-Area Network Level: Once a device is compromised, an attacker can compromise other similar devices that are nearby, and build a botnet of edge devices that can be used to execute a coordinated attack against the grid. For example, an attacker controlling a large number of smart meters can order them to all disconnect and then reconnect at the same time, creating a large load shift that can be disastrous for grid equipment [143].

At the Wide-Area Network Level: This scenario is the most concerning for utility operators, since an attacker could use a compromised edge device as a gateway to *any* of the devices on a utility's SCADA network, leaving substations, control centers, and potentially even generators open to attack. (Our conversations with industry insiders indicate that flat, unsegmented networks are frequently encountered in the power industry—for example, a recent attack on Ukraine's power grid involved "issuing commands directly from a remote station" [70]. Even if networks had some sort of segmentation in place, there would still need to be holes for legitimate communication, which can be leveraged by an attacker as well.) The consequences of such an attack could be disastrous: A 2014 *Wall Street Journal* article declared that malicious attackers could cause a nationwide blackout by taking down fewer than ten critical substations during a period of high demand on the grid, and that such a blackout "could plunge the country into darkness for weeks, if not months" [116]. Thus, a compromised edge device could give an attacker access to one or more of these critical substations, and thereby allow them to cause damage that extends far beyond the loss of a single device.

Given these potential consequences, protecting edge devices is a high priority for utilities.

1.2 The Grid Defender's Dilemma

As a first step towards protecting these edge devices, we developed Autoscopy Jr. [99],[4] a host-based intrusion detection system that used control-flow integrity to sense the presence of rootkits installed on the device. Autoscopy Jr. worked first by learning a profile of "normal" behavior that occurred on the system, and then by monitoring the system and identifying when the system deviated from this profile. It also minimized its burden on the host by living directly within the OS itself, as opposed to using a resource-intensive virtual machine to isolate our code from the kernel. Our prototype was so successful that we were able to successfully transfer the technology to Schweitzer Engineering Laboratories, who incorporated Autoscopy Jr. into their product line.

From Autoscopy Jr., we expanded our scope to look at the larger problem of device *tampering* when Schweitzer proposed the idea a few years later. (Details on "The Schweitzer Scenario" can be found in Section 3.4.) While protecting devices from tampering is a long-standing, well-researched problem (Kent [65] provides one of the earliest examples, and Weingart [151] provides a comprehensive summary of attacks and defenses), SCADA networks present a unique challenge for security professionals, which we define as the *grid defender's dilemma*. A summary of the dilemma is given below (see Chapter 4 for details):

- SCADA networks are vulnerable to malicious attacks with devastating consequences (for example, widespread outages [116]), but are also exposed to a large number of "non-malicious" tamper events, ranging from technician visits to large-scale natural disasters.

- Unlike traditional IT networks, *SCADA networks prioritize availability over everything else, including over system integrity.* This means that correctly identifying and reacting to an event is critical:

[4] We would be remiss if we did not mention Ashwin Ramaswamy, whose original Autoscopy work [100] provided the foundation for our Autoscopy Jr. system.

  - Under-responding to a malicious event gives attackers an opening to execute a major attack and bring the grid down for a prolonged period.
  - Over-responding to a benign event, however, can lead to unnecessary technician visits, device replacements, and service outages.

- The cost of improper responses can be staggering: A single "truck roll" to a remote site costs an average of over $400 [126], meaning that the cost of false-positive responses could add up quickly. On the other hand, however, a large business served by Pacific Gas and Electric could lose over half a million dollars from just a 4-hour power outage [109].

- As a real-time system, grid SCADA networks operate under tight timing requirements [56], and the number of legacy devices in the grid place it under strict resource constraints as well (see Section 2.2 for more details). This means that regardless of what responses are taken, they have to happen quickly.

- Finally, while grid defenders have a clear idea of their security goals and the attacks they want to guard against, their resources are limited: They may not have the time nor the training data to configure a complex protection system for their network.

In total, a SCADA protection system has to operate on embedded devices and should be able to properly identify the events currently affecting the network, determine the correct responses to these events, and execute these responses quickly. Current intrusion and tamper protection systems do not meet these requirements:

Current protection systems lack the power and/or context to differentiate between important tamper events. Many tamper protection systems are host-centered, and simply cannot collect the data needed to detect distributed or context-sensitive events.

Current protection systems treat any sort of tampering as malicious. Even if a system could tell the difference between different events, it often still lacks the capability to change their response accordingly.

Current protection systems have either no response or a single response. Either the systems are detection-only, or they have a single "catch-all" response.

Current protection systems are reactionary. By the time these systems detect an attacker on the network, the attacker is already inside the security perimeter and executing their attack plan.

Current protection systems require a lot of manual configuration. Trying to build and configure some of these systems takes far more time and resources than a grid defender has to spend.

Current protection systems cannot adhere to the grid's inherent performance constraints. These systems are designed for different networks with different goals, and are not able to operate under the restrictions of a grid SCADA network.

To solve this dilemma, we require a protection solution that is flexible and accurate enough to handle different types of tamper events, powerful enough to enact the proper responses to these events, fast enough to operate even under the demands of grid networks, and simple enough to capture a defender's intuition about the problem without placing an undue burden on them.

1.3 Our Proposal: TEDDI

To address the grid defender's dilemma, we propose taking a *distributed* approach to tamper detection, separating the different components (tamper sensing, decision-making, and enforcement) into different entities that can live in different places in the SCADA network.

We drew inspiration for our system from the XACML policy language, which was designed to offer "a method for handling a distributed set of policy components, while abstracting the method for locating, retrieving and authenticating the policy components" [89].

Our tamper protection proposal, which we have named TEDDI (Tamper Event Detection on Distributed Infrastructure), consists of three main components:

Tamper Information Points (TIPs) (Section 6.4)**:** Sensor-equipped[5] programs that live near edge devices and scrutinize their surroundings for potential tamper events.

Tamper Decision Points (TDPs) (Section 6.5)**:** Regional decision-making engines that live in higher-security areas of the network (for example, inside substations), listen for tamper reports from the TIPs it serves, and use the current state of the network to make a fully-informed tamper decision.

Tamper Enforcement Points (TEPs) (Section 6.6)**:** The programs responsible for enforcing the decisions made by TEDDI by enacting defender-defined response sequences corresponding to the decisions.

These components work together to make and enforce tamper decisions as follows (see Figure 1.1):

- Monitors placed near an edge device look for the presence of indicators in the local environment that may comprise a tamper event. As these indicators appear and disappear, the monitors report the indicators' presence or absence to the edge device's TIP.

- The edge device's TIP will do as much event detection as it can with the information it collects, and if it is able to definitively detect an event, it sends that event decision

[5]Note that we use the terms *sensor* and *monitor* interchangeably in this thesis to describe the mechanisms that actually look for indicators in the environment. We started by exclusively using "sensor," but switched to monitor to synchronize our terminology with Bohara, Thakore, and Sanders [18].

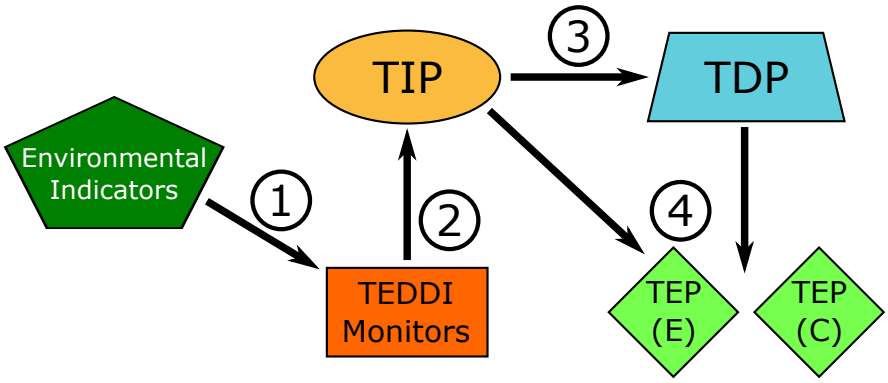


Figure 1.1: A diagram of how the components in our TEDDI system interact. When an indicator appears in the environment around an edge device (for example, a box is shaken), it is captured by monitors near the device (1) and sent to the device's TIP (2), which then attempts to make a tamper decision on its own. If the monitor data is not enough to make a definitive event decision, the TIP requests assistance from its TDP (3), which takes data from a number of TIPs and can make an authoritative decision based on the regional indicators it sees. Once a decision is made by either a TIP or TDP, it is sent to appropriate *edge* and *central* TEPs (4), which can then coordinate the proper response. For more details on this process, see Section 6.

to the *edge* and *central* TEPs assigned to that device.[6] The TEPs can then coordinate the execute the proper responses.

- If the TIP requires data from other devices to make an event decision, it sends an alert to its assigned TDP, which collects this data from all of its managed TIPs and can make an informed, authoritative event decision. The TDP can then send this decision out to the appropriate TEPs.

We selected *factor graphs* [42] as our decision-making algorithm due to its combination of computational power and conceptual simplicity. Factor graphs are bipartite graphs made up of *variable nodes* and *function nodes*, where function nodes define the relationships between different variable nodes (see Section 2.3.5 for more details). In comparison to more common techniques such as Bayesian networks (BNs) and Markov random fields (MRFs), factor graphs are powerful enough to represent any relationship that either a BN or MRF can, and can depict complex BNs and MRFs in a simpler manner by combining

[6] See Section 6.6 for more details on why two TEPs are assigned to each TIP.

multiple variable relationships into a single factor function [23]. As we demonstrate in Section 9.3, we are also able to process factor graphs quickly enough to satisfy the grid's tight constraints.

We began by building an initial TEDDI prototype, which consisted of a single TDP managing five TIPs and ten TEPs, to confirm the feasibility of our idea. Once this step was completed, we constructed the *TEDDI Generation Tool*, which eases the burden of configuring TEDDI on grid defenders by creating the custom TIP, TDP, and TEP programs needed for any arbitrary SCADA network. The tool takes in two key sets of inputs:

- The events, indicators, and monitors that make up TEDDI's factor graph, as well as the responses available for each event.
- The topology of the SCADA network in question, including the types of devices involved (i.e. which devices should be protected with a TIP, and which devices are candidates for hosting a TDP) and the relevant networking information (IP addresses, ports, etc.).

With this information, the generation tool can create the code for the required TIPs, TDPs, and TEPs. To further ease the burden on grid defenders, we included the following features in our tool:

Factor-Graph Domain-Specific Language (FGDSL): We give grid defenders a simple way to define the data and relationships involved in our factor graph by providing a domain-specific language (DSL) [124] called FGDSL, or *factor-graph domain-specific language*. The grid defender defines the relevant events, indicators, monitors, and responses in FGDSL, and the generation tool does the rest, i.e., compiling the data down into a full factor graph, creating a series of logical predicates to represent the graph sequence, and eventually generating the actual code.

Response Suggestion Engine: To assist grid defenders in choosing the right response for an event, we provide a response suggestion engine that offers advice on how to modify the provided factor graph and response sequences. If an event has an especially-long indicator sequence, the tool will suggest adding *pre-events* that detect when an event is about to occur, allowing the system to take a pre-emptive response. Additionally, if two events have similar event sequences but different response sequences, the system will recommend taking responses from one event and applying them to the other.

Network Topology Uploader: We constructed a network topology uploader based on Weaver et al.'s Cyber-Physical Topology Language (CPTL) [149]. The uploader takes a JSON file representing a SCADA network from the grid defender and parses the relevant information, including which devices are connected, which devices need protection via TIPs, and which devices are eligible to host a TDP.

TDP Placement Tool: Once the TIPs are placed, the generation tool uses the TDP Placement Tool to determine i) the best places to put TDPs within the network, and ii) which TIPs to link to those TDPs. The placement algorithm starts by using a greedy Set-Cover algorithm [30] to place TDPs near large clusters of TIPs, and then performs a breadth-first search to link any stray TIPs that were not linked by greedy Set-Cover.

To evaluate TEDDI, we measured its event detection accuracy, factor graph processing performance, and the amount of time and effort needed to configure TEDDI using its generation tool. In each case, we found that TEDDI was either comparable or better than current state-of-the-art SCADA protection solutions:

- Our TIP detected the correct event in 99.2% of our test cases, while our TDP achieved perfect accuracy over 200 rounds of testing.

- Our TIPs processed a 99-node factor graph in just under 233 μs, while our TDP did so in just over 143 μs.
- Our generation tool allows a grid defender to build an event sequence directly from a simple description of the problem, and does not require the extra time or information that other similar systems do.

Through our evaluations, we definitively demonstrate that TEDDI solves the grid defender's dilemma:

- TEDDI's distributed information-gathering and decision-making capabilities allow it to differentiate between important and non-important tamper events.
- TEDDI's granular response strategy gives a grid defender the flexibility to craft the proper response for every event they care about.
- TEDDI's detection strategy can intercept an attacker earlier in the kill chain [54], and potentially keep them off the network entirely.
- The TEDDI Generation Tool makes building and configuring TEDDI easy for any arbitrary SCADA network.
- Our performance measurements indicate that TEDDI can operate efficiently enough to handle even the strict constraints of the power grid.

1.4 Contributions

We make five contributions in this thesis:

- We introduce and define the grid defender's dilemma, and show how the grid's exposure to a wide variety of tamper events, its strict time and resource constraints, and its

focus on availability make this a much more challenging problem than dealing with tamper protection in a traditional IT network.

- We collect and present a set of edge device tamper events based on attacks from both current literature and real-world examples, and build a taxonomy of tamper event types from these examples.

- We propose TEDDI, a novel tamper protection system that addresses the above issues and solves the grid defender's dilemma. TEDDI's distributed setup allows us to capture wide-area events and differentiate them from local ones, its response controls ensure defenders can execute the proper response for every event, and its novel application of factor graphs allow it to operate under the grid's strict constraints.

- We implemented TEDDI, and through our evaluations we demonstrate that TEDDI is faster, more accurate, and required fewer resources to run than current state-of-the-art protection systems. TEDDI's factor graphs can be processed quickly enough to stay out of the way of an edge device's primary task, our event decision mechanisms provide an improved layer of protection against attackers while minimizing false positives and negatives, and our generation tool allows grid defenders to quickly and easily configure TEDDI for their networks and capture their intuitions about the events they are concerned about.

- We demonstrate how TEDDI detects every event in our collection of edge device tamper scenarios, and show how TEDDI solves the grid defender's dilemma.

- Finally, we show how existing computer science solutions can be adapted via TEDDI to solve important problems in power grid systems. Our use of techniques such as factor graphs [42], CPTL [149], and the greedy Set-Cover algorithm [30] are critical to TEDDI's use and operation, and make TEDDI uniquely suited to solve the grid defender's dilemma.

1.5 Thesis Outline

We structure the rest of this thesis as follows: Chapter 2 provides important background information on edge devices, tamper protection systems, and data fusion algorithms; Chapter 3 offers a taxonomy of tampering attacks, along with some example tamper scenarios and a list of common defense techniques; Chapter 4 provides a detailed description of the grid defender's dilemma; Chapter 5 discusses the related work in both tamper and intrusion protection and why it falls short of solving the dilemma; Chapter 6 describes TEDDI's components and how they work together to make decisions; Chapter 7 discusses the TEDDI Generation Tool and how it simplifies the process of building a TEDDI system for grid defenders; Chapter 8 demonstrates how TEDDI addresses some of the tamper scenarios from Chapter 3 and solves the grid defender's dilemma; Chapter 9 evaluates TEDDI for its speed, accuracy, and required resources; and Chapter 10 offers our conclusions and maps out a direction for future TEDDI research.

All chapters are based partially on my prior publications from SECRYPT [102] and ICMC [101]. Parts of Chapters 1, 2, and 5 are based on the Autoscopy Jr. work from my masters thesis [99].

Chapter 2

Background

In this chapter, we summarize the overall layout of the power grid, discuss the types of devices being introduced to the smart grid, describe some basic concepts about tamper protection, present a primer on factor graphs and discuss why we chose them over other fusion algorithms, and briefly mention the XACML origins of our proposed framework.

2.1 The Power Grid: A High-Level Overview

First, we introduce the basic infrastructure of the power grid, and trace how power flows through it. (We stress that this is a high-level summary with few details, and point the reader to other resources, such as Grigsby's book [46], for more details.)

Generation: This group encompasses the power plants that first generate the electricity. For the most past, this process involves spinning a large generator using steam from a turbine, with the power behind the turbine coming from a conventional source such as coal, natural gas, or a nuclear reactor [20]. Other fuels, ranging from sunlight [86] to water [20], are also commonly used.

Transmission: Once electricity is generated, it is distributed to local electrical networks via the transmission system. We define transmission systems by two important criteria:

- The *voltage level* of electricity passing through the wires. Normally, the voltage of electricity leaving a power plant is between 15 to 25 kilovolts (kV) [63], but before entering the transmission system, the power is routed through a transformer that steps up the voltage into the hundreds of kilovolts (generally between 138 and 765 kV) [139]. The reason for this increase is to improve the efficiency of the system, as higher voltages reduce the amount of power lost during delivery [20].[1]
- The *distance* that the electricity will travel. The exact distance power can travel on a single line depends on the voltage involved: "High voltage" lines (100 to 230 kV) have a maximum range of roughly two hundred miles, while "extra-high voltage" lines (235 to 800 kV) can transport electricity four to five hundred miles [63].

While high-voltages are more efficient for electricity transmission, they pose a significant safety hazard, and are much more expensive to insulate from conductive materials [143]. Therefore, once electricity gets close to its destination, its voltage is stepped back down (typically to 10kV or less) as it passes to the distribution side of the grid [20].

Distribution: The distribution portion of the grid covers the "last mile" between individual homes/businesses and the transmission system. Electricity enters this portion of the grid through a distribution substation, which reduces the voltage back down to safer levels and uses electrical buses to route the power in multiple directions [20].

[1] Interested readers are encouraged to consult HowStuffWorks [53] for more details on why high voltages are more efficient.

As opposed to the transmission portion of the grid, distribution systems generally send power between twenty and thirty miles [63].

Once electricity reaches an individual house, it passes though a final transformer that reduces the voltage down to the 120/240-volt service we expect,[2] gets recorded by a power meter, and eventually reaches the requesting appliance.

What makes the distribution section of the grid so important from TEDDI's perspective is that this section is where we see the vast majority of edge devices being installed. We take a closer look at these devices and their attributes in the next section.

2.2 The Smart Grid: Intelligence at the Endpoints

A number of intelligent electronic devices are being installed as part of the smart grid movement [99]. Some examples of these devices include:

- Smart electric meters, which allow utilities to collect usage data without having to send out a technician to read the meter [14].
- Demand response technologies (for example, in-home price displays) that allow consumers to adjust their electricity consumption based on real-time power costs [44].
- Synchrophasors, which give grid operators an up-to-date view of the grid by delivering precise, timestamped power system data to them [144].
- Digital relays, which use signal processing algorithms to detect electrical faults and protect important grid equipment [156].

Intelligent electronic devices in the grid are subject to three important constraints:

[2]Note that this final value can vary from country to country, depending on their standard. Worldstandards.eu provides a complete standard list [155].

Resource Constraints. The specification and capabilities of these devices can vary widely depending on their intended purpose, but in general they have fewer resources than general-purpose computing systems by any measure. Motorola's ACE3600 Remote Terminal Unit, for example, only boasts a 200 mHz microprocessor, 16 MB of Flash memory, and 32 MB of DRAM [82].

Time Constraints. While these devices may be resource-constrained, they are still expected to do their jobs very quickly to accommodate the real-time demands of the grid. Delivery windows for certain types of data are very tight—for example, system protection data related to phasor measurement units must be delivered to its target in at most 30 ms, and breaker tripping commands used to protect power lines must be delivered in at most 8 ms [56].

The installation of smarter edge devices (i.e. smart meters, recloser controls) introduces a third restriction:

Network Constraints. Simply put, "power utilities monitor and control the power grid through a partially unsecured slow legacy network" [122]. This constraint means that the devices and programs using this network must be judicious about the data they send, since they may not have the time or resources to send a lot of information, and any information they do send inhibits other programs from sending their own messages.

Despite these constraints, utilities are installing more and more edge devices in their SCADA networks, with the goal of making their operations more efficient. While smart meters [35] and demand response interfaces [44] are the most well-known examples, this set also includes lesser-known devices such as recloser controls [113].

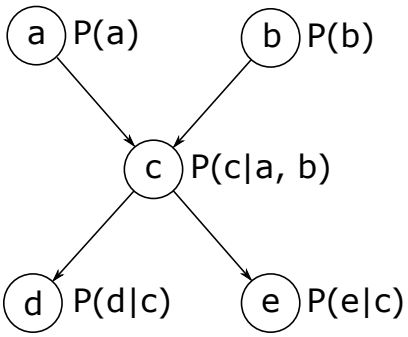


Figure 2.1: A Bayesian Network built from the example in Section 2.3.1.

2.3 Data Fusion Algorithms

In order to make event decisions, we need a way to fuse together the data we collect from our monitors. We selected *factor graphs* [42] as our fusion algorithm, but there are a number of other algorithms available; we outline a few here, and then discuss the reasons behind our factor graph choice.

2.3.1 Bayesian Networks (BNs)

A Bayesian network is a probabilistic model that represents dependencies between variables using a directed acyclic graph [23]. Given a set of variables $S = \{s_1, s_2, \ldots, s_n\}$, we construct a BN graph $G = (V, E)$ as follows:

- For every variable s_i, we add a node v_i to V.
- If the chance of v_i occurring is affected by v_j, then an edge $v_j \rightarrow v_i$ is added to E.

For example, consider the set of Boolean variables $\{a, b, c, d, e\}$, where the $P(c)$ is dependent on $P(a)$ and $P(b)$, and both $P(d)$ and $P(e)$ are dependent on $P(c)$. Our Bayesian network for this set would look like Figure 2.1, with directed edges from a to c and b to c indicating that a and b influence c's value, and similar edges added from c to d and c to e.

BNs are a popular method of modeling variable dependencies, and are seen in a number of protection systems (for example, [105], [141], [146]). However, they suffer from two major flaws: they are unable to capture complex dependencies between events due to their inherent independence assumptions, and adapting them to cover these dependencies can make these models infeasibly complex [23]. This issue would be a big problem for our factor graphs, as our indicators have presence, order, and time components that would all need to be modeled using separate variables. Therefore, BNs lack the flexibility and simplicity we want for TEDDI.

2.3.2 Markov Random Fields (MRFs)

A Markov Random Field is a probability measure over a graph in which a local attribute of any vertex is completely determined by the same local attribute of its neighbors [66]. More formally, if the vertices in a graph $G = (V, E)$ have an attribute a, and we define a_v as the attribute value for a vertex v and $N(v)$ as the set of v's neighbors in G, then the function P is an MRF if $P(a_v|a_{V-v}) = P(a_v|a_{N(v)})$.

If we reconsider our variable set $\{a, b, c, d, e\}$ from before, we can build an MRF equivalent to the BN in Figure 2.1. To do so, we define the function $\phi(w, x_1, x_2, \ldots, x_i)$ to represent the conditional probability function $P(w|x_1, x_2, \ldots, x_k)$.[3] Figure 2.2 depicts our final MRF, which has a 3-variable clique to depict the relationship between a, b, and c, and 2-variable cliques to show c relates to both d and e.

MRFs offer a useful advantage over BNs by allowing groups of vertices in which each pair within the group has an edge between them[4] to have their behavior defined by a single function, avoiding the need to make causal assumptions about variables such as in a BN [23]. However, because MRFs deal with the maximum-sized cliques, it may not be

[3] We adopted the ϕ notation from Frey [42] and Cao et al. [23].

[4] Note that this would be a *strongly connected component* in a directed graph; MRF graphs, however, are undirected.

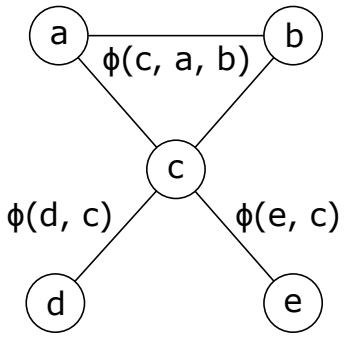


Figure 2.2: A Markov Random Field equivalent to the Bayesian Network from Figure 2.1.

able distinguish group-wide relations from those that only involve a few of the nodes in the group.

2.3.3 Binary Decision Diagrams/Branching Programs (BDDs)

Binary decision diagrams (BDDs), or *branching programs*, are directed acyclic graphs used to represent complex Boolean functions [4]. Nodes are either designated as query nodes, which represent a single Boolean variable and have two outgoing edges representing 0 and 1, or output nodes, which are sink nodes that are labeled 0 or 1 [115]. Given an arbitrary assignment of values for a set of Boolean variables, we can simply traverse the graph based on the assignment of the variables in each query node until we reach an output node, thereby giving us the result of the Boolean statement.

These programs can also be optimized as *read-once branching programs*, where we are only required to check the value of each variable at most once. Unfortunately, we are unable to optimize our own graphs into a read-once branching program, as our graph's indicator sequence setup (which include both indicator order and the time windows they must appear in) means that nearly every node in the graph must be treated as a unique variable.

While we can potentially alter a branching program to handle non-binary outputs, capturing complexities beyond whether or not a variable is present is a tricky endeavor, and would involve creating a new variable to represent relationships like 'Did X occur before

Y?' While setting up such a program is possible, we would prefer a simpler representation of our indicator sequences.

2.3.4 Custom Algorithms

Some protection systems forgo formal methods altogether, and instead develop their own custom system to handle event detection. Oftentimes this involves observing known, accepted behavior and using it as a model to verify future behavior, such as with SCADA-Hawk's snapshots [123] and Amilyzer's flow-matching system [14]. While such techniques can be effective, they have a high risk of false positives due to their learning phases, as any legitimate behavior that does not appear during this phase will be later flagged as suspicious. The cost of avoiding this problem is a prolonged learning period, which is time that grid defenders may not be able to spend.

2.3.5 Factor Graphs

Formally, a factor graph is a bipartite graph that connects a set of nodes V that represents system variables with a set of nodes F that represents functions relating these variables [42]. If a function is dependent on a variable, an edge is added to the graph between the nodes that represent this variable and function.

If we go back to our simple example from Section 2.3.1, we define a variable node for each member of our set $\{a, b, c, d, e\}$, and then define a function node for each conditional probability. We then add edges to show which functions relate to which variables—in this case, we relate $P(a)$ to a, $P(b)$ to b, a and b to $P(c|a, b)$, $P(c|a, b)$ to c, and so forth.[5] The final result is show in Figure 2.3.

The key to a factor graph's flexibility are the functions represented by F: These func-

[5]Note that while we use directed factor graphs for TEDDI to better depict the input and output variables for function nodes, factor graphs can be either directed or undirected [42].

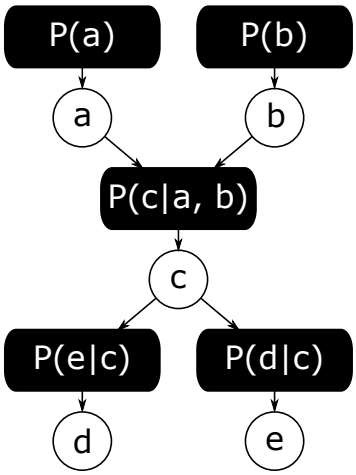


Figure 2.3: A factor graph equivalent to the BN and MRF shown in Figures 2.1 and 2.2, respectively. Variable nodes are shown in white, while function nodes are show in black.

tions can be any "arbitrary factorization of the joint distribution" [42]. This feature allows factor graphs to model any distribution that can be modeled by a BN or MRF, but do it using a simpler model, as its functions "can simplify a complex BN or a complex MRF by reducing the number of functional relations that have to be defined" [23]. (This statement applies to branching programs as well, as it is trivial to model whether a variable is set to 0 and 1 within a factor graph.)

For example, while Figure 2.3 may seem a bit cluttered compared to its BN and MRF counterparts, it offers significant benefits for TEDDI, which looks for specific indicator sequences that denote events:

- Rather than having to define a variable for every single piece of a relation function (Is y present? Was x present before y? Did y appear too long after x did?) as we would be forced to do in a BN, we can represent the entirety of x and y's relationship in a single graph node.

- If a relationship only involves z variables, we can define a function node that only involves those variables. This allows us to explicitly depict how variables relate to one another, rather than obscuring relations between groups of variables as in an

MRF.

Therefore, using a factor graph gives us the flexibility to relate our variables in almost any manner we like. For these reasons, we chose factor graphs as the fusion algorithm for TEDDI.

Factor graphs have been applied to a number of different fields in the past, including signal processing [73], robot navigation [57], evaluating trust on ad-hoc networks [128], and even detecting malicious users in a system [23]. To the best of our knowledge, however, TEDDI is the first system to use factor graphs for physical tamper protection.

2.4 XACML and the Power of Distributed Systems

At a high level, XACML is a specialized XML-based language designed for use in creating and enforcing access control policies [31]. The goal of XACML is to provide "a common language for expressing security policy" [91] to assist companies and other groups in managing the security policies of every system operating on their network [91]. Of most importance for our purposes is one of the requirements laid out by version 3.0 of the XACML standard:

> *"To provide a method for handling a distributed set of* ***policy*** *components, while abstracting the method for locating, retrieving and authenticating the policy components"* [91]. (emphasis added by original author)

To that end, the standard borrows a pair of terms from RFC 3198 [152] for its decision and enforcement entities, and adds a third term to represent sources of policy attributes:

> *"Policy Decision Point (PDP)...A logical entity that makes policy decisions for itself or for other network elements that request such decisions"* [152].

> *"Policy Enforcement Point (PEP)...A logical entity that enforces policy decisions"* [152].
>
> *"Policy information point (PIP)...The system entity that acts as a source of* ***attribute*** *values"* [91]. (emphasis added by original author)

For TEDDI, we have co-opted the PDP, PEP, and PIP terminology and purposes for our tamper decision/enforcement/information points, which are all discussed in more detail in Section 6. We made this choice because tamper events may not be distinguishable simply by looking at the local environment of a single device. For example, a single box shaking could signal that an attacker is trying to break into a device, but a lot of boxes shaking simultaneously could signal a larger event such as a earthquake. The responses to these events would be very different: An active attacker might merit a severe response, while an earthquake might cause us to suspend our security procedures to allow workers from others utilities to help repair the damage. Taking a distributed approach to tamper detection, therefore, helps us to make better, more-accurate event decisions and improve the overall security and availability of the grid.

2.5 Network Intrusion Protection Systems (NIPS)

Although TEDDI is a tamper protection system, it bears a striking resemblance to a network intrusion protection system (NIPS). (We discuss this similarity a bit more in Section 6.7, but the general idea is that packet sequences for a NIPS are analogous to indicator sequences (Section 6.3) for TEDDI.) Below, we offer a brief description of NIPS in general, and discuss general attacks and defenses in this space.

NIPS generally work by monitoring traffic as it passes through their network, and analyzing this traffic to look for suspicious activity [96]. Many of these system use signatures as their primary detection method, and look for either specific series of packets or specific

data within those packets that indicate a potential attack. While a NIPS lacks the device-level information that a host-based protection system collects, its network-centric setup allows it to detect “attacks that involve low-level manipulation of the network, and ...easily correlate attacks against multiple machines on a network” [96].

Some of the common attacks against NIPS are:

Sequence Insertion: A packet is crafted such that the NIPS accepts it, but the target host does not, disrupting the packet sequence that the NIPS sees and allowing the attack to go through [96].

Sequence Evasion: A packet is crafted such that the NIPS rejects it, but the target host accepts it [96]. Evasion can be done in several different ways, depending on the configuration of the NIPS:

- The attacker can wait out the NIDS [114]. This can be done by fragmenting the attack packets, and then sending them with just enough time in between each one so that the NIPS does not reassemble the fragments, but the target host does.
- The attacker can wait out the targeted host [114]. An attacker can do this by fragmenting a set of benign packets, sending them to the target host, and waiting until the host discards them. If the benign fragments are still being monitored by the NIPS after the host throws them away, the attacker can then send maliciously crafted packet fragments that match those still held by the NIPS, fooling the NIPS into thinking the packets are harmless. The malicious fragments will then reach the targeted host, and the attacker can now simply the remaining pieces of the malicious packets—the NIPS will think these fragments match future segments, but the host will combine them and discover the malicious payload.

- If a router lives between the NIPS system and the target host, the attacker can use the time-to-live (TTL) field of the packet to evade detection [114]. Here, the attacker splits their malicious packet into several fragments, and creates a benign copy of one of the fragments (with a TTL field set to expire when it reaches the router between the NIPS and the target. When the attacker sends their packet fragments (with the benign fragment sent it place of its malicious twin), the NIPS is fooled into think the packet is not a threat, but the target receives a incomplete packet set (thanks to the router dropping the benign fragment copy), and waits for the final piece of the packet. The attacker now graciously provides the remaining malicious fragment, which sneaks past the NIPS (it thinks the fragment is part of a whole new packet) and completes the packet set held by the target.

Despite these attacks, NIPS remain very popular among network defenders, and several mature systems are commercially available (for example, Bro [21] and Snort [121]).

2.6 Autoscopy Jr.

Finally, we offer a few words about the work that preceded TEDDI, Autoscopy Jr. [99], and how this work eventually inspired the TEDDI project.

Autoscopy Jr. is based on Kprobes [76], a kernel tracing framework built into Linux that allowed a user to pause execution and inspect the current state of the OS, including register values and kernel memory contents. For Autoscopy Jr., the goal of this inspection is to see whether the kernel's control flow is being altered unexpectedly, i.e. modified by a rootkit looking to hide important information. The systems accomplishes this task by examining the return address on the call stack when an important function is called, and seeing where the current function was called from.

Autoscopy Jr. starts in a *learning mode*, where it monitors the kernel to see what its various control flows look like under normal operation, and is eventually switched to a *detection mode* to check future behavior against its learned model. If the observed return address of a function leads us to an unexpected location that did not appear during our learning mode, Autoscopy Jr. reports this issue back to the user.

In addition to its avoidance of a virtual machine, Autoscopy Jr. also offers a *probe profiling tool* to further avoid overburdening an edge device. The profiling tool allows a user to measure the amount of system overhead imposed by individual probes, and remove probes that are interfering with the device's primary tasks.

The response to Autoscopy Jr. was incredibly positive, and we managed to successfully transfer the technology to Schweitzer Engineering Laboratories (SEL) for use in their own products.[6] However, the system falls short when considering the full grid defender's dilemma:

- Its host-centered design leaves it unable to detect distributed events happening in the larger network. (In fact, as an intrusion detection system, it neglects tamper events entirely.)
- It detects problems, but does not have the ability to respond to them.
- Building a model of allowed behavior via the learning mode may take more time and resources than a grid defender can spare.

To solve this dilemma, we will need to broaden our scope beyond a single device, and build a system that can detect and respond to the events we are concerned about.

[6]This also established the partnership that eventually led to SEL returning to us several years later with the tamper problem that led to the creation of TEDDI.

Chapter 3

A Taxonomy of Tampering

In this chapter, we discuss the different types of physical tamper attacks that an edge device may face, and highlight some examples of these attacks found in the literature and the real world. Based on the nature of these attacks, we group them in four categories: device data accesses, device additions, device modifications, and device replacements. We also highlight some of the defense techniques used to detect and prevent tampering.

3.1 Device Data Access

This first category of tampering attacks does not involve altering the device's operation at all—rather, the attacker simply tries to access secret information stored on the device for future use. Some example techniques of this attack include:

- Placing passive probes on the circuitry of a device to record the signals that appear [151].
- Inserting a hardware Trojan that monitors and transmits the inter-chip communications of a device [39].

- Cutting through the sensor-mesh-encasing potting material that covers sensitive device components, which can be accomplished using lasers, chemicals, sandblasters, or even a knife in the hands of a careful individual [151]. (This sort of attack was one of the prime protection focuses of the IBM 4758 [119].)
- Imprinting data on a memory chip using radiation, low temperatures, or high voltages [151].

Edge devices in the power grid, which could possess secret cryptographic keys for communication or house sensitive customer demographic and usage information, are prime targets for these techniques. Based the above techniques, we can construct the following tamper attacks:

Simple User Data Heist: An edge device stores a batch of customer power usage data in a non-volatile memory chip, and periodically sends this data to a utility control center. To access this data, an attacker performs the following steps:

1. First, the attacker opens the case of the device.
2. Next, they place probes on the output pins of the memory chip.
3. Finally, they read out the data when the device sends it to the utility.

Complex User Data Heist: This is similar to the above attack, except that the data in question is stored in volatile memory, and the attacker does not want to wait around for the device to send it to the utility. To obtain the data, the attacker uses an imprinting attack as described by Weingart [151]:

1. The attacker starts by bombarding the device with X-rays (or other suitable radiation), permanently burning the data into the chip.
2. The attacker then cracks open the device's case to access the memory chip.

3. Rather than probing the chip, the attacker simply removes (and potentially replaces) it

Device Credential Heist: An edge device stores its encryption key in a memory location that is physically secured by a sensor mesh encased in a special potting material, just as in the IBM 4758 [119]. To gain access to the key, an attacker must do the following:

1. Pry open the case of the device.
2. Find a way through the mesh surrounding the key storage memory.
3. Probe the chip to extract the key material.

3.2 Device Additions

The second category of tampering attacks involve malicious additions to the device that make it behave in unexpected ways. Here, we highlight two specific tamper attacks:

Pin-In-The-Meter Attack [132]**:** In 2010, an outlet reported that one consumer stopped their (mechanical) meter in the following way:

1. They drilled a hole in the bottom of the device.
2. They stuck a pin through the hole, which kept the wheel inside the meter from spinning.

Maintenance Mode Attack [33]**:** After two months of receiving anomalous readings from some of its meters, a Brazilian utility discovered that their devices had been victims of a sophisticated tampering scheme:

1. The attackers had first opened the device and broken a tamper-evident seal underneath it.

2. Next, they installed a microcontroller attached to an RF receiver inside the meters, allowing them to remotely send signals to the meters and switch them into a special mode that "disconnects the meter without interrupting the electricity flow to the consumer" [33].
3. Finally, the attackers made a convincing replica of the tamper-evident seal on the device, making it appear as if the meters had not been altered.

What makes the maintenance mode attack even more interesting is that legitimate technicians go through the same process (minus the additional hardware) when performing regular maintenance—in fact, this is what the special mode was intended for in the first place! Defending against this attack, therefore, may require additional information than just the attack steps.

3.3 Device Modifications

Our next group of attacks, and perhaps the most common one, is the actual modification of the software and/or hardware of an edge device to bend it to the attacker's will. Some examples in this space include:

- Unauthorized alteration to a meter's firmware, causing it (and in a worst-case scenario, many others simultaneously) to disconnect from the grid [14].
- The Ukraine power grid cyber attack, in which attackers loaded malicious firmware onto the utility's serial-to-Ethernet converters, deleted the master boot records of workstations to render them inoperable, and even reconfigured one of the utility's backup power supplies to cause it to disconnect soon after the attack began [70].
- The Maroochy Water Services attack, where a rogue ex-employee disabled alarm

systems and generally altered the system "so that whatever function should have occurred at affected pumping stations did not occur or occurred in a different way" [3].

- The Stuxnet worm, which modified the code on programmable logic controllers to sabotage the centrifuges they were running, and which featured infected USB sticks as one of its delivery methods [40].

While many of these are software modifications, and therefore outside of TEDDI's scope, our goal is to detect when the attacker has physically accessed the box, *before* they have the chance to modify the device's software. We highlight two potential physical tampering scenarios:

Malicious USB Firmware Update: In this scenario, a malicious third party with legitimate credentials (such as a contractor hired by the utility) takes the following steps:

1. They access a cabinet-enclosed edge device using a key.
2. They open the cabinet and remove the plugs blocking the device's USB port(s).
3. Finally, they plug in an infected USB device intended to upload a malicious firmware update. (What the update actually does to exploit the system is less important, as it is the exploit delivery process that TEDDI focuses on.)

Once again, we note that this event has a benign counterpart in which a utility technician uploads a legitimate software update to the system, which means that we need to verify whether or not a firmware modification is expected or not.

Return-To-Debug Attack: This scenario is based on Weingart's description of how ion beams can be used as a tampering tool [151]. The attack is as follows:

1. The attacker removes the cover of an edge device.

2. Using an ion beam, the attacker alters the circuitry in the device by connecting the debug pins on the device's circuit board, granting the attacker access to key storage locations on the device.

From here, the attacker can access sensitive data on the device, and potentially modify this data to cause to malfunction or misreport its state.

3.4 Device Replacements

Finally, we consider the fourth category of tampering attacks, where an edge device is removed from the network and is either replaced by a malicious device or removed to be used and/or modified for malicious actions at a later date. We define the following scenario for this category:

The Schweitzer Scenario: This scenario is the one that originally inspired the TEDDI project. In the summer of 2013, while I was interning at Schweitzer Engineering Laboratories (SEL) for the summer, one of the group's leaders came to us with a troubling scenario involving the vandalizing of some of their edge devices. The attackers went through the following steps:

1. They used a tool (either a crowbar or hammer) to break the lock off of the cabinet containing a device.
2. They opened the cabinet using the newly-unlocked door and proceeded to vandalize the device, although we received no details as to how the devices were manipulated.

While this attack was more of an annoyance than a major problem, members of the Schweitzer product team worried what might have happened had the attackers taken one more step:

4. Disconnecting the edge device from its network access point, and connecting the attacker's own device (for example, a laptop pre-loaded with exploits to use against other devices) in its place.

Given the structure of some SCADA networks, this fourth step could allow attackers to reach deeper into the network to attack higher-value targets.

The Sensor Subversion Scenario: This scenario is a slight variation of the Schweitzer Scenario, in which an attacker knows that the above sequence is being used to protect a device. To subvert this sequence and access the edge device, the attacker attempts the following:

1. The attacker starts by drilling a small hole in the cabinet containing the edge device.
2. Next, the attacker inserts a tube into the hole and squirts glue onto the cover switch [117], causing it to seize up and not move when the cabinet door is opened.[1]
3. The attacker then proceeds as they would in the Schweitzer scenario, opening the cabinet and unplugging the edge device.

There are more ways that an attacker could get into the cabinet—for example, they could cut a hole through the side with a torch, and bypass the cover altogether—but the overall event structure is similar to the two described above, so we do not discuss it here. Additionally, tamper events involving device thefts can also be considered, with the difference being that the device is unplugged, but not replaced.

[1]While this possibility was not brought up by our Schweitzer contacts (and thus was not included in the Schweitzer Scenario), we discuss it here to illustrate the wide variety of tamper scenarios the grid faces.

3.5 Non-Malicious Tampering

The four types of tampering we discussed so far were cases of malicious tampering. While we mentioned that a few malicious attacks have benign counterparts (such as firmware updating), the grid faces a large number of non-malicious tamper events that affect its availability. (In fact, there has been some debate over whether squirrels actually pose a bigger threat to the grid than hackers [92], and trees have played a role in a substantial number of blackouts, such as a large 2003 blackout in Italy [12] and an August 1996 blackout on the U.S. West Coast [50].) To cover the realm of accidents, mistakes, and natural disasters, we present the following examples of non-malicious tampering:

The Taum Sauk Reservoir Overflow [62]**:** We examine a structural failure at the Taum Sauk hydroelectric facility, where water is pumped from a lower reservoir into an upper one during periods of low electricity demand, and released back into the lower reservoir to spin turbines and generate electricity during peak demand periods [80].

In December of 2005, the facility's water pumping system overfilled the upper reservoir, causing that reservoir's dam to fail and dump the reservoir's contents onto the surrounding countryside [62]. From the post-failure investigation report, we see that the problem stemmed from the pressure transducers that were used to measure the dam's water level:

1. Some of the pipes covering and protecting those transducers had broken free and moved from their initial positions.
2. The movement of the transducers caused them to report water levels lower than the true level of the reservoir.
3. The errant water level values misled the pump system into overfilling the reservoir.

Earthquakes: Natural disasters are interesting events to investigate because they are often wide-spread events, and thus require information from a number of distributed sources to detect. As an example, consider the following earthquake scenario:

1. When an earthquake strikes, each individual edge device in the area experiences severe shaking at roughly the same time.
2. Each device must then gather information from others around it to confirm that the event is wide-spread, and not an isolated local incident. The latter scenario may indicate that a malicious tamper event is beginning, while the former leads us to conclude that an active attacker is not involved (and furthermore, we may want to initiate a utility's disaster response protocol to prepare for potential damage/outages).

3.6 Tamper Protections

People have been trying to keep malicious actors from tampering with important data for centuries, as demonstrated by the ancient practice of using hot wax and signet rings to both seal and authenticate documents [22]. This problem carries significant weight in the digital world, as many important identity assertions, such a digital certificates, rely on the ability of a party to keep sensitive data secret. Thus, we must store such data in a place with sufficient tamper protections to keep adversaries from accessing or altering it.

However, the term "tamper protection" is rather vague, and can mean different things to different people. Smith and Marchesini summarize the thinking in this space:

- "Tamper resistance: *It should be hard to penetrate the [tamper protection] module.*
- Tamper evidence: *Penetration attempts should leave some visible signal.*

- Tamper detection: *The device itself should notice penetration attempts.*
- Tamper response: *The device itself should be able to take appropriate countermeasures when penetration is detected"* [117].

Existing tamper protection schemes fall into one (or more) of these four categories. The race between the developers of these schemes and the attackers who try to defeat these schemes has become a cat-and-mouse game, and technologists have developed a number of sophisticated attack and defense mechanisms over the years. Weingart offers a comprehensive summary of these tactics [151]; we highlight some of his defense categories below:

Barriers: Materials that need to be penetrated to reach the inner logic of edge devices. Examples range from conventional materials (such as metal and brick) to special coatings that protect chips on a circuit from probing.

Sensors: Monitors that look for environmental phenomena that indicate a potential attack. Examples include voltage sensors, temperature sensors, probe sensors, radiation sensors, accelerometers, and meshes of wire sensors wrapped around a device.

Seals/Switches: Mechanisms that leave evidence when tampering occurs. This category not only covers tamper-evident solutions like paints, labels, and packaging, but also microswitches that can detect vibrations or small movements.

Data Destruction Methods: Methods of destroying data such that the adversary cannot recover it, such as overwriting data on RAM chips or physically destroying the memory and/or device.

In recent years, even policymakers have waded into the tamper protection field. In 1994, the National Institute of Standards and Technology (NIST) released Federal Information Processing Standards Publication 140-1 (FIPS PUB 140-1) [134] to provide a standard for cryptographic modules that could be used by the United States government. These

guidelines were updated in 2001 by FIPS PUB 140-2 [135], which is the most recent official version of the standard (although draft versions of FIPS PUB 140-3 [136] have been released).

The FIPS standards outline four security certification levels that modules can obtain based on the amount of protection they provided, with one being the lowest and four being the highest, and listed the features that modules needed to reach each level:

- Level 1 provides no special guidance for physical security beyond mandating the use of "production grade equipment" [135].
- Level 2 requires either tamper-resistant or tamper-evident controls on the module, such as "tamper-evident coatings or seals or...pick-resistant locks" [135].
- Level 3 adds tamper detection and response controls to the list, mandating that the mechanisms used "have a high probability of detecting and responding to attempts at physical use, access, or modification" [135].
- Finally, Level 4 adds environmental testing requirements to ensure a module is not compromised by operating outside its specified voltage and temperature ranges, and demands that the provided detection and response controls "provide a complete envelope of protection around the cryptographic module" [135].

Obtaining Level-4 certification is extremely difficult: According to NIST, only twelve such certifications has ever been awarded, and only two have been awarded since 2003 [83]. An example of such a module is the IBM 4758 [119, 120], which was the first ever module to obtain Level 4 certification under these guidelines. The 4758 design included a grid of conductors wrapped around the device to detect physical tampering of the device, a set of temperature sensors to ensure the device remained within its operational limits, and a series of hardware "ratchet" locks that restrict software access to important data once the ratchet reaches a certain level [118]. The design has held up remarkably well over

time; attacks against the 4758, such as Clayton and Bond's attack on IBM's Common Cryptographic Architecture [28], focus on flaws in the applications running on the 4758 rather than the hardware itself. To the best of our knowledge, there are no known successful attacks against the device's physical and software configuration security. IBM has produced several iterations of its cryptographic coprocessor since the debut of the 4758; the current model on the market is the 4765 [55].

Chapter 4

The Grid Defender's Dilemma

With all the time and effort spent on tamper protection research in the past, why have current solutions not found their way into the power grid? We believe the issue lies with the uniqueness of the grid environment and the sheer diversity of scenarios and threats the grid must take into account, giving rise to a problem we call the *grid defender's dilemma*. In this chapter, we describe in detail the issues that underly the dilemma, and list the reasons that current protection solutions cannot fully address it.

4.1 What Is The Dilemma?

Fundamentally, the grid defender's dilemma boils down to the tension between the integrity and the availability of the grid's SCADA network—more specifically, the struggle between keeping attackers off the network and keeping the lights on. We give a more detailed explanation below:

- First, grid SCADA networks are open to attacks that could have severe, long-lasting consequences. For example, an attacker could cause a widespread, long-lasting power outage by taking down critical substations during a period of high demand

on the grid [116]. Protecting against these types of attacks, therefore, is an absolute necessity.

- However, edge devices within the grid have to deal with a number of different types of tampering, not all of which are malicious or even involve an active adversary. Some examples include:

  Technician Visits. Broken and malfunctioning devices are a fact of life in the power industry, and utilities often have to send technicians out to their remote sites to "tamper" (update, repair, replace, etc.) with their equipment.

  Natural Disasters and Weather Events. Earthquakes, hurricanes, and other such disasters often wreak havoc on a utility's infrastructure, and may damage or destroy their edge devices.

- For standard IT networks, administrators err on the side of integrity, as the consequences of a network breach are far more significant than a network outage, and thus most tamper protection systems are geared towards this order of priorities. In the power grid, however, these priorities are inverted: Availability is the top priority, as the critical nature of these networks dictates that they need to be up and running as much as possible, even in the face of malicious attacks.

- This focus on availability places a huge burden on both grid defenders and any protection solutions they use, because:

  – Selecting the proper response to a tamper event affecting the grid is absolutely critical. Under-responding to a malicious event leaves the grid open to a major attack, but over-responding to a benign event could lead to unnecessary technician visits, device replacements, and service outages. The cost of unnecessary responses can be staggering: TekTrakker estimated that sending a technician on a single trip to a field site would cost a utility an average of over $400 [126],

and a study of Pacific Gas and Electric's customers found that a 4-hour power outage would cost a small/medium-sized business nearly $5,000, while a large business would lose over half a million dollars [109].

  - Correctly identifying a tamper event that is affecting the grid becomes important as well, since defenders have little chance of choosing the correct response if they do not know what event they are dealing with. This also means we must be specific in our identification; just reporting that "a device is being tampered with" is not enough.
  - The real-time nature of the grid means responses may need to be selected and executed quickly, so there may not be time to query a human for input. In these cases, the system should be able to select the proper response and enact it quickly without outside intervention. Ideally, an event would be detected and responded to early enough that if a malicious attacker is present, he or she does not gain access to the SCADA network at all.

- Finally, grid defenders are busy people, and while they may have a rough idea of the threats they face and the events they want to detect, they may not have the time or resources to build and configure a complex protection system.

4.2 Why Haven't We Solved The Dilemma?

Despite the large amount of research in tamper protection, current state-of-the-art solutions suffer from several problems that keep them from solving this dilemma:

Lack of Context Awareness: Current protection systems lack the power and/or context to differentiate between important events. A large number of tamper protection systems are focused on protecting data on a single host, and thus fail to account for a large class of tamper events simply because they do not gather the appropriate contextual

information. For power grid networks, however, being able to detect these events is vitally important, as it may make the difference between distinguishing a malicious event from a non-malicious one. If an edge device is shaking, is it because of an active adversary, a natural disaster like an earthquake, or is it just sensor noise generated by a vibrating appliance or a passing car? If an edge device's cabinet is breached, is the perpetrator an attacker trying to exploit the device, a utility technician trying to fix the device, or a colony of bees just looking to build a hive? Most protection systems simply do not collect enough information to answer these questions.

Lack of Tampering Awareness: Current protection systems treat any sort of tampering as malicious. Part of the response to the above points is to simply declare any sort of tampering as malicious, and respond accordingly. The problem, however, is that these lone responses serve as "catch-alls" for whatever tamper event they detect, so they are naturally geared towards covering worst-case scenarios like malicious attacks. The grid's focus on availability and the presence of non-malicious tamper events, however, makes this approach infeasible, because a strong response will be overkill for most situations, and may end up doing more harm than good. For example, reacting to a service technician as if they were an attacker could lead to added costs and decreased system availability, while a severe tamper reaction to a natural disaster could slow down recovery efforts unnecessarily.

Additionally, certain responses are ruled out by the grid defender's dilemma, regardless of the severity of the event. For example, the IBM 4765 "zeroizes its critical keys, destroys its certification, and is rendered permanently inoperable" [55]. Such a response makes reducing the grid's availability easy: Simply poke and prod the device until it destroys itself!

Lack of Response Granularity: Current protection systems have either no response or a single response. Sadly, system with single catch-all responses are better positioned

than the majority of tramper protections systems, which have no response mechanism at all and thus are left wide open to a malicious attacker.

Lack of Timeliness: Current protection systems are reactionary—that is, they are reliant on looking for activity that indicates a malicious actor is on the network, at which point the actor is already putting their attack plan into effect. Given the importance of grid SCADA networks, operators would prefer to stop the attack earlier in the "kill chain" [54] and try to keep malicious actors off their network completely. While it is impossible to accomplish this goal 100% of the time, an earlier response could still limit the amount of damage an attacker can do.

Lack of Appropriateness: By "appropriateness," we mean the ability of a solution to do its job without interfering with the edge device's primary tasks, and current protection systems cannot adhere to the grid's inherent performance constraints. Exceeding the time and resource constraints from Section 2.2 means keeping a power edge device from doing its job, which is exactly the situation we want to prevent. Systems designed for a traditional IT setting are simply not equipped to meet these demands.

Lack of Automation: Current protection systems require a lot of manual configuration. For example, the Response and Recovery Engine [161] requires that the operator construct a full attack response tree (ART), complete with every potential step taken by an attacker and the responses that correspond to each step, for each security goal they want to maintain, while SCADA-Hawk [123] must run for a prolonged period to collect data and capture the behavior snapshots it needs for anomaly detection. Grid defenders, however, have neither the time nor the expertise to build these sorts of systems.

These problems open the door for a different type of tamper solution, one that is flexible and accurate enough to handle different types of tamper events, powerful enough to enact

the proper responses to these events, and simple enough to capture an operator's intuition about the problem without placing an undue burden on him or her.

Chapter 5

Related Work

In this section, we discuss some of the literature on tamper protection schemes, and how they differ from our TEDDI proposal.

5.1 Tampering vs. Intruding

Before we begin, we need to address the similarities between tamper detection and its closely-related cousin *intrusion detection*. Both fields have received extensive attention over the years, and occasionally the question of what constitutes a tamper detection system or an intrusion detection system (IDS) is just how the systems are labeled. (A good example of this is SCADA-Hawk [123], which operates much like an anomaly-based IDS despite being labeled as a tamper detection program.) While we consider work from both disciplines as relevant to our own research, we differentiate the fields as follows:

- Tamper protection focuses on preventing both unauthorized *physical* access to a device and unauthorized changes to a device's *structure.* For example, a system that prevents malicious actors from changing a line of code through the device's local interface or modifying the internal circuitry of a device would be a tamper protection

system.

- Intrusion protection focuses on preventing both unauthorized *network* access to a device and unauthorized changes to a device's *behavior.* For example, a system that prevented attackers from remotely exploiting a device vulnerability or feeding the device bad data or operational parameters would be an intrusion protection system.

We can break these fields down further into five major categories, as shown in Figure 5.1:

Software Tamper Protection: Programs that prevent unauthorized access/changes to a device's software.

Hardware Tamper Protection: Programs that prevent unauthorized access/changes to a device's hardware.

Signature-Based Intrusion Protection: Programs that protect against known examples of bad behavior that are taken either by or against a device.

Anomaly-Based Intrusion Protection: Programs that protect against devices that deviate from a known or learned set of "normal" behaviors.

Hybrid Intrusion Protection: Programs that can use both signature- and anomaly-based techniques to protect devices.

By these definitions, TEDDI falls into the Hardware Tamper Protection category. However, TEDDI also shares a striking number of characteristics with a signature-based intrusion protection system, which we discussed previously in Section 2.5.

In addition to the categories above, there are also some defense types that do not fit nicely into any of them; we discuss some of these in Section 5.7.

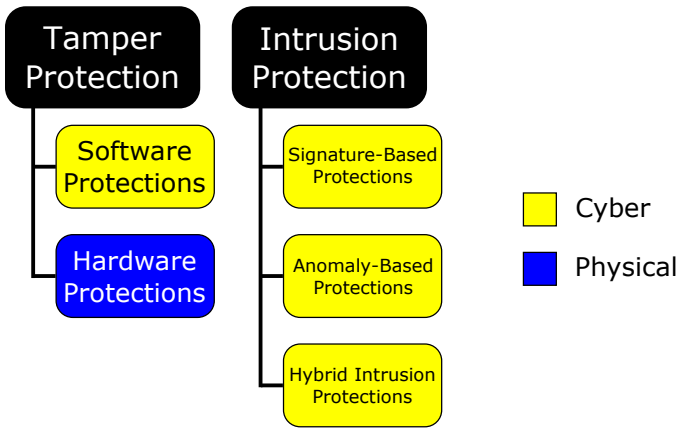


Figure 5.1: A taxonomy for prior work in both tamper and intrusion detection. Note that tamper protections are split between cyber and physical attacks, while intrusion detection systems focus primarily on the cyber domain.

5.2 Software Tamper Protections

Software tamper detection receives a lot of attention in conjunction with digital rights management, and a number of tools have been released (such as packers like UPX [140]) to make software harder to understand and thus harder to modify. Obfuscation, or altering a piece of code to make it harder to understand and/or reverse engineer, has also received considerable attention in the academic world: Neves and Araujo integrate techniques like overloading directly into a C++ compiler to allow it to obfuscate compiled code automatically [85], while Collberg et al. take a more dynamic approach by constantly changing the arrangement of code on the client, exhausting the attacker's ability to adapt and keep pace [29]. Anckaert, Sutter, and De Bosschere take this approach to the extreme by making every particular copy of a program unique, including updates, and ensuring that other modified copies of the program are not usable [8]. (While this defense is primarily geared towards software piracy, this would hinder malicious, security-specific tampering as well, as each individual copy of the software would need to be tampered with in a different way to achieve the attacker's malicious goal.) Finally, Okhravi, Riordan, and Carter cross over into the hardware tamper protection realm by evaluating the effectiveness of changing the hardware and/or operating system behavior on the fly, and determining where the benefits

of dynamic changes lie [90].

Another popular method of tamper detection is self-verification, where a program runs through a series of tests to determine if its code has been modified inappropriately. Often this process is accomplished through checksums, where a message (or piece of code, in this case) is run through a hash function to create a special value that is hard to duplicate with a different (i.e., erroneous or maliciously-crafted) input [117]. For example, Giffin, Christodorescu, and Kruger use self-checksumming, augmented with a special self-modifying-code mechanism, to detect and defeat code modification attacks [43], while Tsang, Lee, and Pun take a checksum-like approach by placing "protectors" at various points in the code and taking an appropriate response (which can vary by protector) if the code has been changed [131]. Chang and Atallah use a set of "guards" embedded in the code to perform certain security tasks, such as checksumming or code repair, that increase the difficulty of modifying the code without authorization [25]. All of these techniques increase the difficulty of tampering with protected programs, as an attacker is forced to create a tampered version of the code that accomplishes their evil task while also matching the original checksum value, an almost-impossible task if the checksum is created using a cryptographic hash function [117].

5.3 Hardware Tamper Protections

Hardware tamper protections are what most people think about when they consider "classic" tamper protections. The roots of this area of research run deep, and includes seminal works such as Kent's ideas for *tamper-resistant modules* [65], White and Comerford's *ABYSS* platform [153], White et al.'s work on the *Citadel* system [154], Tygar and Yee's *Dyad* platform [133], and Smith, Palmer, and Weingart's work on the *IBM 4758* [119, 120]. Today, a number of hardware solutions are available commercially, either in the form of trusted platform modules ([9, 51, 58]) or cryptographic coprocessors [55]. However, these

solutions are geared towards single-device protection, and may have trouble operating in the extreme conditions power devices must face (for example, the IBM 4765 is only designed to run within a temperature range of 10 to 35 degrees Centigrade, and treats any temperature outside that range as a potential attack [55], while the SEL-651R recloser control must be able to run within a temperature range of -40 to 55 degrees Centigrade [110].

In recent academic work, Dragone uses several layers of "patches" connected by a wire mesh and laid out in a random pattern such that anyone attempting to drill through the material could not avoid hitting a patch and triggering a response [37], while Megalingam et al. discuss connecting a smart meter's power supply to the screws that hold its case together, rendering the meter useless if anyone tries to open the case and access its internals [78]. Desai, on the other hand, takes an obfuscation approach by adding extra dummy states to a chip's finite state machine, and only transitioning to the true functional states if a special code-word is provided to the chip [36].

Physically unclonable functions (PUFs) have also received a lot of attention as a protection technique, as they can verify the integrity of a device or piece of code in a way that is difficult for an attacker to replicate. For example, if a user produces a message authentication code (MAC) for a piece of software using the output of a PUF as a key, if an attacker wishes to tamper with the software, he or she is forced to either replicate the PUF's secret or find a key/malicious code pair that produces the same MAC, neither of which is an easy task. Much of the work in this space focuses on improving PUFs themselves, such as Niewenhuis et al.'s work on using scan chains as a low-cost PUF [88], Maiti and Schaumont's proposal to improve the quality of ring-oscillator-based PUFs [75], and Rührmair and van Dijk's analysis (and critique) of "Strong PUF" proposals [107]. Other work highlights the many ways PUFs can be used for security, including Bolotnyy and Robins's research on using PUFs to improve the security of RFID systems [19] and Suh, O'Donnell, and Devadas's processor architecture that combines PUFs with several off-chip memory protections to provide a private and secure execution environment [125].

Finally, hardware Trojans can been viewed as a tamper protection scheme with the traditional roles reversed, as now the attacker is the one trying to protect device hardware (or at least their malicious addition) from being circumvented by defenders. Here, both sides are on the lookout for tampering, with attackers trying to protect the sanctity of their additions and defenders looking for ways to verify that the hardware they receive is the hardware they expect, so analyzing both sides can provide some insight into tamper protection and detection. On the offensive side, malicious actors use a number of techniques to avoid being discovered, such as use rare or distributed trigger conditions to avoid exposure during testing [16]. Meanwhile, on the defensive side, some recent protection schemes include fingerprinting chip designs using path delays [60], inserting dummy flip-flops to circumvent triggering mechanisms [108], using linear programming to examine the physical properties of logic gates and detect the presence of extra circuits [95], using capacitor behavior as a detection mechanism [39], and combining a number of known detection techniques to reduce the chance of a Trojan hiding and activating [16].

5.4 Signature-Based Intrusion Protections

Signature-based detection methods have taken a lot of heat in recent years, as they provide little protection against new and novel attacks, or even older attacks that have undergone small changes [129]. Nevertheless, the topic remains an active area of research. Litty, Lagar-Cavilla, and Lie's *Patagonix* system studies the behavior of a system's hardware to identify programs that are running (and which may be hidden from the operating system) and verify the programs against a trusted set of binary hashes [72]. Jiang, Wang, and Xu's *VMWatcher* reconstructs the workings of a potentially-compromised OS from the safety of a virtual machine, allowing an antivirus program or other protection device to analyze the untrusted OS without fear of being compromised [59]. However, as pointed out in my prior intrusion detection work [99], both of these programs are based on the use of a virtual

machine, which is not feasible for use on resource-constrained embedded systems.

Azab et al.'s *TrustZone-based Real-time Kernel Protection system (TZ-RKP)* makes use of ARM's secure TrustZone environment to build a system that is isolated from untrusted programs yet still able to effectively protect the kernel from a malicious attacker [10]. From within TrustZone, TZ-KRP can evaluate the impact of running specific pieces of code on the system and block them from harming the system if necessary, while also restricting user processes from accessing kernel memory directly. However, strict enforcement of these rules may cause problems with legacy programs that rely on this functionality, and its host-centric design may have trouble gathering information on regional events.

Grochocki et al. study a number of potential grid attacks and determine the optimal intrusion detection system to combat them: A centralized intrusion detection system coupled with sensors embedded in the remote devices [47]. While this approach is very similar to our own proposal, the sensors in this case are focused on internal values such as "health reports, firmware and software integrity, and memory contents" [47], rather than the state of the device's physical environment.

Zhao, He, and Yao's *Filter driver and Proxy based Website Anti-tamper System (FPWAS)* system applies a distributed approach to the problem of protecting websites, using a file monitoring system and a set of web proxies to protect against a set of known website attacks [159]. While the proposal is labeled as a tamper protection system, its behavior more closely resembles an IDS looking for known bad behavior—for example, scanning files on the server for illegal modifications. However, given the number of additional proxy servers required by the system, deploying this system would be a costly proposition.

Zonouz et al.'s *Security-Oriented Cyber-Physical State Estimation (SCPSE) system* combines intrusion detection alerts with power system information to more accurately estimate the security state of an electrical network [162]. SCPSE builds an attack graph that traces the possible paths an attacker could follow via exploiting network nodes, and deter-

mines how power information in the system should be correlated to devices in the network. In its monitoring mode, SCPSE uses power flow information and intrusion alerts to estimate the attacker's path through the graph, and thus reveal which devices in the network are potentially compromised. Despite its increased awareness, however, this system still ignores external environmental factors that might affect the network's security state, and does not incorporate possible responses into its design.

Roblee, Berk, and Cybenko's *Process Query Systems (PQS)* combine host and network monitoring to determine which nodes in a network might be compromised [105]. Process sensors at the host level report information back to PQS's fusion engine, which uses conditional probabilities to relate events to one of its attack/failure models. While the system focuses on bad behavior within devices and their network, the system could potentially be configured to handle environmental events as well. However, trying to track a large number of behavior models may produce a prohibitive amount of overhead on an edge device in the grid.

Wang and Hauser's *evidence-based trust assessment (EBTA) framework* tries to evaluate the trust they can place in a device based on the evidence they collect [146]. The authors collect a series of data vectors within a small time window, define a loss function that captures the consequences of taking an action a when the device's trust level is t, and finally use a parameterized risk function to decide whether the device is trustworthy. However, the program suffers from the weakness of its Bayesian basis, and only makes a binary trust decision, which is not granular enough to extend to intrusion response (and such an extension would make the system too unwieldy and complex to use, as it would need to make a separate trust decision for each tamper event we care about).

Cheetancheri et al. propose a coordination system for local detection programs targeted at detecting worm outbreaks within a network [26]. When a device receives an alert from its local IDS, it sends a message to m randomly-selected other devices in the network. If

a recipient has also received an alert from its own IDS, it combines the message with its own data and forwards the message to another m randomly-selected nodes. If a message chain is forwarded enough times (i.e., enough devices are reporting an intrusion), a global outbreak message it sent to all of the devices. The major issue with this approach is that large portions remained ill-defined: What responses (if any) are taken in response to the global warning? How does the IDS this system relies on detect worm attacks?[1] Without more clarity, its applicability to the grid defender's dilemma is unclear.

Valdes and Skinner's *Probabilistic Alert Correlation (PAC) system* uses Bayesian-based data fusion as a way to reduce false positives within an intrusion detection system [141]. The system maintains a list of *meta alerts* that might represent an attack, and adds individual alerts to a meta alert if the system thinks they are similar. The system also maintains a minimum similarity threshold and a priority field within the meta alert, all with the goal of showing the administrator only the issues that are most likely to be security violations. The similarity metric, however, can become cumbersome as the number of meta alerts and alert features grow, and attack class similarity is evaluated via a static matrix that would require updating as different threats emerge.

The *Response and Recovery Engine (RRE)* takes the concept of attack signatures a step farther than most systems: Instead of simple signatures, the RRE uses *attack response trees* (ARTs) to define the security goals it wants to maintain, the various ways these goals might be violated, and the possible responses that could be taken to maintain those goals [161]. As intrusion alerts come in, the system determines which nodes in the tree have been reached, which represents what the attacker has achieved thus far. The trees are converted to Markov decision processes, which are then solved to determine the optimal action to take against the attacker. The RRE also has local and global components, which allow it to monitor both the state of individual boxes and the overall state of the network. While the RRE

[1] We assume that this is based on a signature-based IDS, as pairing it with an anomaly-based IDS would cause problems with making sure every device saw the same anomaly.

is arguably the system that most closely resembles TEDDI within the current literature, however, it currently does not consider external events such as environmental factors, and the response trees have to be complete enough to cover the entire attack space that the administrator is concerned about.

5.5 Anomaly-Based Intrusion Protections

Another popular method of intrusion detection involves verifying device actions against a pre-defined or learned model of system behavior, and raising an alarm when the system deviates from its model. There are two common flavors of this type of system:

Specification-Based Intrusion Detection: The model of system behavior is derived from a known specification or protocol. Specification-based systems have become very popular for SCADA network protection, as SCADA networks are fairly static and tend to exhibit predictable behavior that rarely changes [27]. In most cases, the restrictions focus on the allowed behavior of the protocols spoken by SCADA devices, including Modbus [27, 45], IEC 61850 [48], IEC 60870-5-104 [157], ANSI C12.22 [13, 15], and even more general protocols such as IEEE 802.15.4 [61].

These specification can also be based on the physical properties of a device, such as with Edwards's hardware-trojan-detecting IDS [39], or designed in accordance with a specific security policy, such as with Riley, Jiang, and Xu's *NICKLE* [104] and Wang et al.'s *HookSafe* [147].

Learning-Based Intrusion Detection: The model of system behavior is learned by the system, either through canned or live instances of normal behavior. This approach is the most popular in standard IT networks, although it often focuses on network traffic to minimize the damage an adversary can cause to the wider network. For example, Cucurull, Asplund, and Nadjm-Tehrani analyze a number of features (packet

rates, packet type ratios, etc.) to calculate the distance any given packet is from a device's "average" packet, and raise an alert if the number of suspicious packets reached a certain threshold [32]. Mehdi, Khalid, and Ali Khayan propose combining Software-Defined Networking technology with standard anomaly-detection algorithms to improve the detection rates of security problems in smaller home or office networks [79].

Kenaza et al. aim to reduce an IDS's false-positive rate via an adaptive support vector data description (SVDD)-based learning approach [64]. While the approach begins with a set of labeled data for training, the system is periodically tested during operation, with the feedback from the test getting fed back into the system to allow it to update its algorithm. Over time, the additional knowledge improves the classification rate of the IDS, reducing false positives while maintaining a similar detection rate to a similar system without the extra learning. However, the system requires expert intervention to provide the necessary feedback, which may not always be available, and the evaluation is focused on software and network attacks rather than tamper events.

Boggs et al. look into aggregating intrusion alerts across multiple entities to detect zero-day attack as the exploits propagate across the Web [17]. Each individual device exchanges information with the others regarding abnormal requests that it receives. The appearance of a similar abnormal request at multiple devices signals that it may be an exploit, and the system alerts an administrator when this scenario occurs.

Düssel et al. propose an anomaly-based detection system to look for new and unknown attacks against critical infrastructure [38]. The system extracts features from the byte sequences inside captured packets, and compares them to its previously-learned "normal" state to look for unusual payloads. Their evaluation produced some impressive numbers, with attack detection rates above 88% and a low false-positive rate of 0.2%. However, this system requires a large dataset up front to learn what normal packets look like, and by the time the system encounters the anomalous packets,

the attacker has already penetrated the network boundary.

The *State Relation based Intrusion Detection (SRID) system* tries to defend SCADA systems for bad data injection attacks by using the implicit relationships between different variables within the system [145]. The system starts by determining how each component of a system influences the others (for example, raising the temperature of a boiler increases the pressure of the steam it outputs). From there, SRID uses these component relationships to look for anomalies that indicate bad data (from the above example, if the boiler temperature drops but the pressure readings remain the same, something is wrong). However, the time needed to analyze the system and learn about component relationships may be time that grid operators do not have.

Ali and Al-Shaer propose an anomaly-based IDS using a model built from event logs collected from smart meters [6]. Noting the predictable behavior of these devices, the system constructs a labeled Markov chain based on the log data it collects, and uses it to verify the future behavior of the meters. However, this system requires a fair amount of data (namely, event logs) to capture the network's behavior, and forces the operator to translate their desired security properties into temporal logic predicates.

Mitchell and Chen's *Behavior-Rule based Intrusion Detection System (BRIDS)* puts a distributed twist on anomaly detection by distributing the monitoring chores among all the various end devices—more precisely, the behavior of every device x in the system is observed and verified by another device y [81]. This relationship is not necessarily one-to-one, as devices with more resources are asked to monitor more devices, which minimizes the burden on the network's resource-constrained devices.

Yoon et al.'s *SecureCore system* examines how multi-core embedded devices could be leveraged for anomaly detection [158]. SecureCore dedicates one or more cores to monitoring the behavior of the remaining cores, validating behavior by comparing pre-developed execution timing profiles of important applications with the program's

behavior at runtime.

Bohara, Thakore, and Sanders take a machine-learning approach to intrusion detection by using unsupervised clustering algorithms to spot anomalous behavior in security logs [18].Their systems looks at logs both from individual machines and from network-wide monitors, and extracts features from these logs to look for data clusters that are indicative of either a denial-of-service attack of the presence of malware on a host. While this system must currently wait until an attack is already taking place before it detects a problem, the system appears to be adaptable to monitoring logs generated by physical or environmental sensors.

Despite its billing as a tamper detection system, we find that Sousan et al.'s *SCADA-Hawk system* [123] more closely resembles a classic anomaly-based intrusion detection system: Despite being one of the few systems to include hardware monitoring in its scope, it looks only for unexpected hardware behavior, and has no mechanism to prevent hardware modification. SCADA-Hawk uses a system of "collectors" (low-level signal monitors) and "agents" (storage programs for collector data) to learn what behaviors are considered normal in different parts of the network. The system can then be shifted into a monitoring mode that looks for behavior that deviates from the learned models.

Like SCADA-Hawk, the "model-based IDS" presented by Roosta et al. [106] also falls under the anomaly-based umbrella. This IDS targets towards protecting wireless process control systems, and much like TEDDI, the system is comprised of both *field* and *central* IDSes, with the former monitoring devices in their own corner of the network and the latter monitoring data from both the field IDSes and external data sources. Like other anomaly-based approaches, however, this system requires enough data to proper distinguish normal and abnormal behavior, which may not be readily available. Also, while responses can be automated, only the central IDS can initiate a countermeasure, as the field IDSes are just passively monitoring their

sensors

Finally, a number of systems rely use control-flow integrity (CFI) as a mechanism for determining when an attacker has modified a system. Van Der Woude [142], Petroni and Hicks [93], and our lab's recent *Autoscopy* work [97, 99, 100] all monitor execution paths within the kernel to learn what behavior is expected and detect anomalous flows. Similarly, Tang, Sethumadhavan, and Stolfo [127] look for anomalous "microarchitectural execution patterns" [127] caused by malware that can be observed via hardware performance counters.

5.6 Hybrid Intrusion Protections

Some intrusion detection systems combine elements of the signature-based and anomaly-based protection methods to protect systems. The two most notable examples of this are the open-source intrusion detection systems Bro [21] and Snort [121], which can be adapted to use the detection scheme that is best suited to the environment. However, both require policies to be expressed in their own special scripting language, and Snort's Active Response feature is limited to simple network actions.

On the academic side, Benmoussa, El Kalam, and Ouahman install both a misuse detection agent and an anomaly detection agent on collaborator networks that serve as early-warning systems for their important networks [11]. When something suspicious is found, these agents report back to a manager agent on the critical network, which allows the network to prepare for similar suspicious behavior within its own borders. While this project's distributed setup and use of packet and log parsers as sensors make it similar to TEDDI, there are two key differences: This system i) currently restricted to detecting software attacks, and ii) intervenes only after viewing suspicious behavior on its collaborator networks. TEDDI, in contrast, attempts to intervene even earlier than this by stepping in the moment someone accesses a device on the collaborator network.

The idea of a *contextual* IDS, or providing external context information about the system to a generic IDS, has gained some attention in recent years. Examples include Hansen's proposal to add SCADA-specific context to assist in judging the intent of an action [49] and Amann et al.'s input framework for incorporating blacklists, malware checks, and other external services into a standard IDS to augment its protection capabilities [7].

5.7 Other Protection Work

One protection technique that does not fit nicely into our taxonomy is "Security-as-a-Service," (SECaaS), where a company simply brings in an outside expert to manage the security of their enterprise. These experts ensure that the company's security tools are always up to date, and they may even bring in lessons learned from their work in other domains to further inform their security posture. Some commercial examples of SECaaS include AlertLogic's ActiveWatch [5], McAfee's cloud-based security services [77], and WebRoot's SecureAnywhere endpoint protection [150].

Lin et al. examine the challenges involved in detecting attacks on cyber-physical systems, propose that combining information from both the cyber and physical domains is key to detecting these attacks, and discuss some potential methods for detecting and responding to attacks [71]. While the authors highlight some of the same concerns that TEDDI raises, such as the difficulty in distinguishing attacks from other physical events, the paper mostly focuses on the impact of malicious commands and data, rather than attacks characterized by physical indicators.

Laszka et al. consider the problem of setting an optimal attack detection threshold for an IDS [69]. They model the process of compromising a device as a game between an attacker and defender, and algorithmically determine the optimal threshold settings based on the costs of IDS false positives and the amount of damage an unmitigated attacker

can do. However, the paper gives few details about whether the IDSes involved here are signature- or anomaly-based.

Another hard-to-categorize system in the CAPMS system, which attempts to detect and respond to cyber attacks against the grid in real time [138]. While the proposal is still light on specific details, the setup is fairly similar to TEDDI: They plan to use a distributed set of nodes to gather information about the network, combine "advanced algorithms with cybersecurity monitoring" [74] to determine the state of the network, and automatically respond to problems to mitigate them earlier than other systems. However, this system does not consider physical tamper events, and does not address either the handling of non-malicious tamper events or the ease of configuring such a system for an individual utility network. Additionally, as it still appears to be in the early stages of development, we cannot definitively determine which category of intrusion detection (signature, anomaly, specification, or something else) that the system falls under.

Emulation-based intrusion detection, where a suspicious program is run in a simulated environment to reveal its behavior, is also a potential protection approach [2]. However, there are a number of evasion techniques against this approach, such as using self-modifying code or simply using esoteric instructions [2], and the time and effort needed to set up an emulation environment and run suspicious code through it is more than a grid defender can spare.

5.8 Prior Work vs. The Grid Defender's Dilemma

With all of the work done in this space, the question arises: Can any of these systems solve the Grid Defender's Dilemma? As shown in Table 5.1, the answer is a definitive "No." Every system we evaluated fall short in one of the key aspects of the dilemma, and many fall short in several categories. Only TEDDI, which we built specifically with critical

infrastructure and the grid defender's dilemma in mind, satisfies all of our criteria.

5.9 Factor Graphs and Security

Finally, we should note that we are not the first to propose using factor graphs as a security mechanism; this distinction belongs to the *AttackTagger* system developed by Cao et al. [23].[2] AttackTagger used factor graphs to discern the state of a user (either benign, suspicious, and malicious) based on the sequence of actions that the user takes. The authors demonstrated how graphs could be used in this manner could be used to detect compromised user accounts, even before the accounts had compromised the actual system, and also discovered several attack sequences that had been previously overlooked. The system does not, however, consider physical tampering in its scope.

[2] In fact, it was this team that originally suggested that we use factor graphs in TEDDI after seeing how well they worked in AttackTagger!

Table 5.1: A sampling of current protection systems and how they fare against the grid defender's dilemma.

System	Handle Distributed Events?	Handle Benign Events?	Flexible Response Setup?	Responds Early In Kill Chain?	Easy To Configure?	Adheres To Grid Constraints?
NA [85]				Yes	Yes	Yes
CMMN [29]	Yes				Yes	
ADSDB [8]	Yes			Yes		Yes
GCK [43]						Yes
TLP [131]			Yes			
CA [25]			Yes			
Kent [65]				Yes		N/A
ABYSS [153]				Yes		Yes
Citadel [154]				Yes		Yes
Dyad [133]				Yes		Yes
IBM 4758 [119]				Yes		Yes
Dragone [37]				Yes		Yes
MKRN [78]				Yes		Yes
Desai [36]				Yes		
PUFs				Yes	Yes	Yes
Patagonix [72]		Yes				
VMWatcher [59]		Yes				
Autoscopy [99]		Yes				Yes
FPWAS [159]	Yes	Yes				
SCPSE [162]	Yes					Yes
PQS [105]	Yes	Yes		Yes		
EBTA [146]	Yes	Yes				N/A
CADLRS [26]	Yes					
PAC [141]	Yes					
RRE [161]	Yes		Yes	Yes		Yes
Edwards [39]					Yes	Yes
KLBS [64]	Yes					N/A
BHSS [17]	Yes	Yes				
DGLBSK [38]	Yes					Yes
SRID [145]	Yes					Yes
CAPMS [138]	Yes		Yes	Yes		Yes
AAS [6]	Yes	Yes				Yes
BRIDS [81]						Yes
SecureCore [158]		Yes				Yes
BTS [18]	Yes					
S-Hawk [123]	Yes	Yes				Yes
RNLV [106]	Yes		Yes		Yes	Yes
VDW [142]						
PH [93]						
TSS [127]						
BEO [11]	Yes			Yes		
TEDDI	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***

Chapter 6

The TEDDI System

In this chapter, we describe the general architecture of TEDDI, and explain each component of the system in detail. We also describe the TEDDI Generation Tool, which we use to create TEDDI systems for arbitrary SCADA networks.

6.1 Problem Assumptions and Attacker Model

Before we get into the details of TEDDI, we first state our assumptions about the problem:

- We assume that the SCADA network is always available, and that packets always reach their intended destinations. Given that utilities require reliable SCADA networks to properly manage and maintain their infrastructure (and any disruption in this service would draw the utility's attention), this assumption appears to be reasonable.

- We assume that the sensors used by TEDDI always report the correct values, and are immune to malfunctioning or manipulation. We believe this is a reasonable assumption because the same physical (i.e., the cabinet or device exterior) would be protecting the sensors as well as the edge device. (We note that our Sensor Sub-

version Scenario from Section 3.4 violates this assumption, but point out that the attacker still has to penetrate the boundary to compromise the cover switch.)

- TEDDI makes one assumption about the topology of the underlying network: There is at least one node present that does not require protection and is able to support a tamper decision point (TDP, Section 6.5). (If the utility has several disconnected networks to maintain, each network requires at least one TDP-eligible node.) Otherwise, TEDDI can operate on any arbitrary SCADA network with minimal configuration.
- We assume that all of our tamper information points (TIP, Section 6.4) are equipped with equivalent sensor sets, and are synchronized to take their sensor snapshot at roughly the same time. While the system will still operate without the latter assumption, it makes calculating the regional tamper states of a TDP more challenging because of the time difference between readings. (The level of synchronization required will depend on how long we wish to wait for data to reach the TDP; we discuss this more in Section 6.5.)

Next, we define the capabilities of the attackers we are targeting as follows:

- We assume that the attacker must go through an edge device to access the SCADA network. More specifically, we assume that the attacker must penetrate some sort of physical boundary, which could be as simple as the device's own exterior, to access the edge device's hardware and network access point.
- The attacker is unable to inject packets into the network without first gaining access to an edge device and plugging a cable into the device's access point. Wireless access points are not considered in our analysis, as they may be reachable outside the device's physical boundary.
- We only consider attacks on edge devices, and not other sorts of power equipment. Tampering with power lines, for example, is considered out of scope.

- Network attacks originating from outside the SCADA network are considered out of scope.
- While TEDDI's sensors monitor the device's physical boundary, we assume that there are no limits to the tools an attacker can use or the time they can take to penetrate this boundary.

6.2 TEDDI Architecture Overview

TEDDI is made up of three components:

- *Tamper Information Points* (TIPs, Section 6.4): Programs that collect sensor data and attempt to make local tamper decisions.
- *Tamper Decision Points* (TDPs, Section 6.5): Programs that take the sensor data from TIPs, determine the regional state of the network area they monitor, and make tamper decisions when asked by the TIP.
- *Tamper Enforcement Points* (TEPs, Section 6.6): Programs that listen for tamper decisions and execute responses based on those decisions.

TEDDI takes a distributed approach to tamper detection by placing TIPs, TDPs, and TEPs all throughout the network it is protecting, which improves its information-gathering capabilities and allows the system to detect regional tamper events that locally-based protection systems, such as the IBM 4758 [119], would miss.

To illustrate how TIPs, TDPs, and TEPs work together to make tamper decisions, consider the following example:

1. A utility operator uses the TEDDI Generation Tool (Chapter 7) to construct a simple tamper system consisting of a single TDP and three TIPs: A, B, and C. The operator

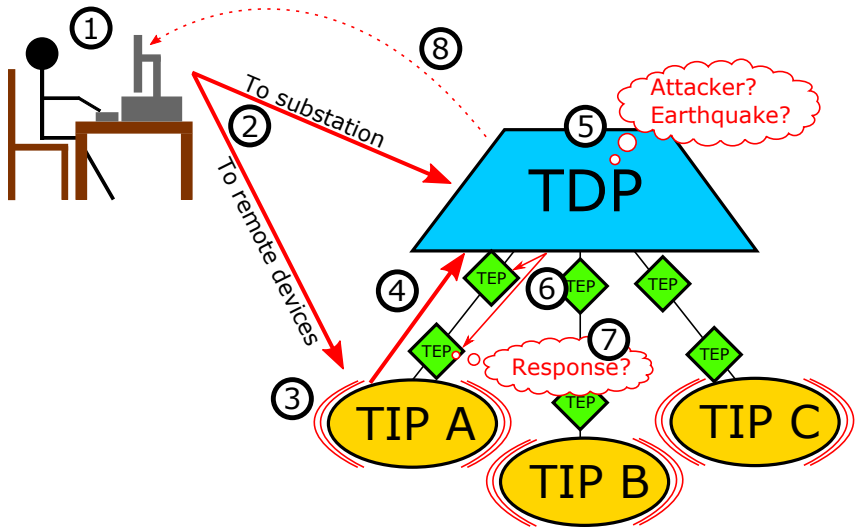


Figure 6.1: A diagram of the example given in Section 6.2. First, a utility operator builds the tamper system using the generation tool (Step 1), and then deploys the various components to their proper locations in the network (Step 2). When a TIP senses shaking (Step 3), it sends an alert to the TDP (Step 4), which then uses it full information base to decide exactly what is happening (Step 5). This decision is then sent to the appropriate TEPs (Step 6), who then decide the proper response to the event (Step 7).

denotes that the TIPs are equipped with an accelerometer as part of their sensor set, and tells the system to watch out for shaking as either part of an attack or an earthquake.

2. The TDP and TIPs programs are deployed, and the TIPs begin monitoring their environment for event indicators.

3. TIP A experiences intense shaking, causing its accelerometer to exceed its threshold. (See Section 6.4 for more information on how TEDDI deals with thresholds for multi-dimensional monitors such as accelerometers.) A's limited factor graph (Section 6.3) knows that either an attack or an earthquake is occurring, but it cannot differentiate between the two possibilities without knowing the state of other boxes in the system.

4. A sends an alert to the TDP about its situation, and requests assistance on making a decision.

5. The TDP receives A's alert and attempts to determine the overall state of the system. In doing so, it finds that B and C have also experienced intense shaking (and in turn, have also sent the TDP alerts about potential tampering).

6. The widespread shaking causes the TDP's full version of the factor graph (Section 6.3) to decide that the shaking is due to an earthquake. This decision is passed along to the two TEPs associated with A.

7. Because the shaking is due to an earthquake, A's TEPs do not take any action for the time being, since the utility does not want to reduce the system's availability unnecessarily. (However, if the shaking had been identified as part of an attack, the TEPs could execute a severe response set—revoking certificates, monitoring traffic, etc.—to ensure an attacker does not gain access to the SCADA network.)

6.3 TEDDI Factor Graphs

At the heart of our tamper decision engine is a factor graph [42] that looks for sequences within its sensor data to determine what event is occurring. By using these graphs, we simplify the TEDDI setup process by capturing the grid defender's intuition about tamper events, and by not requiring a large amount of data or configuration to operate the system.

Our graph, as shown in Figure 6.2, is constructed from two important datasets:

Events: The set $E = \{e_1, \ldots, e_j\}$ of tamper events we want to detect.

Indicators: The set $I = \{i_1, \ldots, i_k\}$ of phenomena connected to the events in E. For example, if E includes the Schweitzer Scenario from Section 8.2, I will need to include an indicator representing an open cabinet door. The presence or absence of an indicator i is calculated by looking at its corresponding monitor m to see if the

monitor reading has reached or crossed an operator-defined threshold. (Our threshold setup may require some pre-processing of sensor data; see Section 6.4 for details.)

Indicators can be classified as *local*, in which their value depends solely on the value of the monitor at the edge device, or *regional*, which depend on the monitor values of all of the TIPs operating under a TDP, and are therefore only visible to TDPs. (How a TDP calculates its regional indicators is discussed in Section 6.5.)

While they are not explicitly included in the factor graph, the system's *monitors* represent a third important set. Monitors are excluded from the formal factor graph because TEDDI assumes a one-to-one relationship between them and the indicators they look for. We make this assumption for simplicity and without loss of generality, as anything that breaks this assumption can be easily modeled by breaking monitors down into separate components or using multiple indicators with different thresholds.

We build our factor graph via a three-step process:

1. For each concerning event e_j, the operator defines a sequence of indicators that signals e_j's presence, as well as the maximum amount of time that can pass between each indicator. This process is described in more detail in Section 6.3.1. In Figure 6.2, for example, we define the sequence "Indicator 1, and then Indicator 3 within W seconds of 1" for Event 1, "Indicator 4, then Indicator 1 within X seconds of 4, and then Indicator 2 within Y seconds of of 1" for Event 2, and "Indicator 5, and then Indicator 1 within Z seconds of 5" for Event 3.

2. The operator ranks the events by their importance, declaring which events are the most important to detect. Here, the events happen to be ranked in numerical order: Event 1 is the most important event, Event 2 is the second-most important event, and Event 3 is the least important event of the three.

3. Finally, the indicator sequences are arranged within the factor graph in order of their

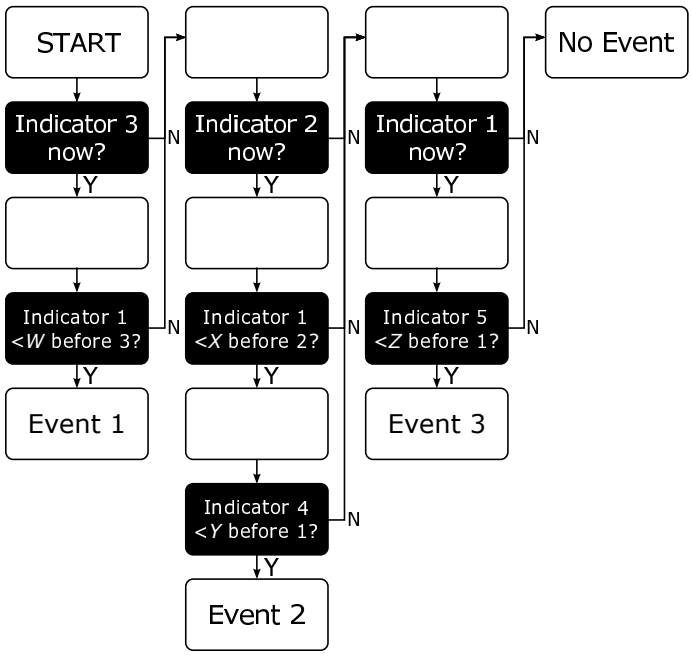


Figure 6.2: An example factor graph generated by our TEDDI system. The blank nodes represent intermediate steps in each event sequence, but are just treated as placeholders between factor nodes in our system. Note that this example looks for the sequences "1, then 3 within W seconds of 1" (Event 1), "4, then 1 within X seconds of 4, then 2 within Y seconds of 1" (Event 2), and "5, then 1 within Z seconds of 5" (Event 3). See Section 6.3.1 for an explanation of why sequences are encoded in reverse.

rank, assuring that events are checked in that order. In the figure, Event 1 will be the first event TEDDI looks for, followed by Event 2, and then by Event 3.

Whenever we poll our monitors for data, we calculate the presence of indicators based on this data, and then use our factor graph to see what events are occurring, starting from the most-important event.

This setup offers two advantages over other fusion algorithms. First, the resulting graph is much less complex than a comparable graph generated using another fusion algorithm. For example, a equivalent graph created using a Bayesian network would require that our factor nodes be split into its ordering and time-window components, leading to an explosion in the potential state space. Second, in comparison to the setup requirements of other systems, such as the complete attack response tree of the RRE [161] and the initial training

period for SCADA-Hawk [123], constructing a factor graph in the above manner is simple and less time-consuming. However, we must raise the following points:

- The sequences themselves *must* be linearized to be properly encoded in the factor graph. For example, a sequence such as "1 *or* 2, then 3" would have to be split into two the separate sequences "1, then 3" and "2, then 3," and then ranked sequentially to ensure they are properly prioritized. This splitting is currently a manual process, and while we assume that a grid defender will have a clear idea of how events should be encoded and ranked, Section 6.7 discusses what could happen if this is not the case.

- Currently, TEDDI only returns a single event decision even if multiple events are present simultaneously, which makes properly ranking the events a critical task. We are reliant on the operator having a clear sense of the relative importance of events, and list in Section 6.7 what could happen if this assumption is violated.

When we generate the final factor graph, we create two versions: a *full* version for tamper decision points, and a *limited* version for tamper information points. We create the two versions to account for differences in the information bases between decision and information points:

- If a sequence contains one or more regional indicators, the sequence is truncated at the *latest-occurring* regional indicator (i.e., the first one that TEDDI will encounter in the sequence) in the limited factor graph. If this sequence truncation causes multiple sequences to appear the same in the limited graph, those events are collapsed into a single sequence with a rank equal to that of the highest-ranked event.

  For example, if Indicator 1 in Figure 6.2 were a regional indicator, then the limited factor graph given to the TIPs would look like Figure 6.3. Any time the graph encounters Indicator 1 when looking through its event sequences, it would have to

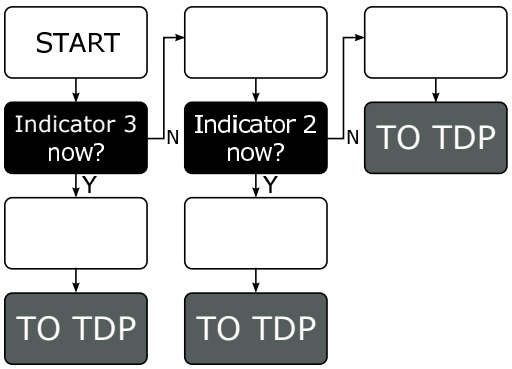


Figure 6.3: A limited version of the factor graph from Figure 6.2, where Indicator 1 is classified as a regional indicator. In this case, since Indicator 1 is part of every sequence, the TIP will not be determine if any of the events it is looking for are present, and will have to request assistance from its TDP every time it reads data from its monitors.

defer its event decision to its TDP, which would have the proper context information to resolve whether or not the indicator is present.

- If an information source is only available to tamper decision points (such as data from an external database), monitors and indicators associated with that information source are not included in the limited factor graph. (Note that the limited graph in Figure 6.3 would look the same if Indicator 1 was an external indicator rather than a regional one, as both types are only resolvable by the TDP)

While TEDDI's use of indicator sequences captures a grid defender's intuition about potential tamper events, the way TEDDI looks for these events is a bit unorthodox. We discuss TEDDI's sequence-checking method in the next section.

6.3.1 How TEDDI Looks For Sequences

People generally define indicator sequences by the order in which the indicators occur: "A first, then B, and then C." However, TEDDI must wait for the entire sequence to appear before making an event decision, and thus its decision is made at the moment in time when the *last* indicator in the sequence occurs. Therefore, we encode the sequences in *reverse*

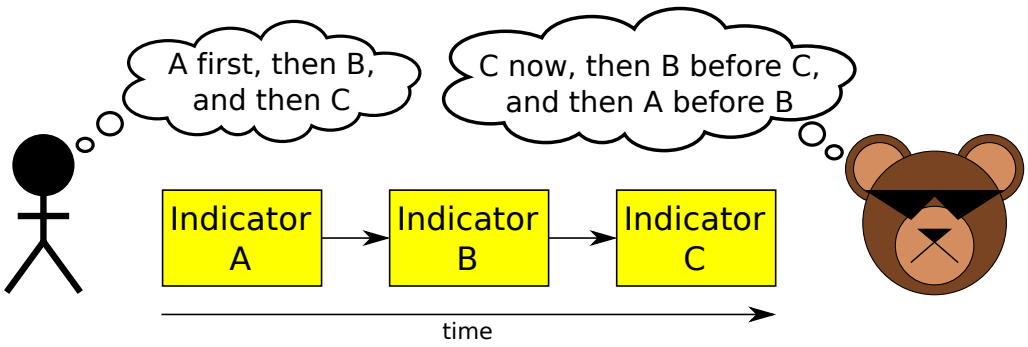


Figure 6.4: A diagram showing how users and TEDDI define indicator sequences in opposite directions. Users normally think of sequences by starting with the earliest indicator and ordering them chronologically. TEDDI, however, has to wait for the entire sequence to appear before declaring that an event is present, so it starts by waiting for the *last* indicator in a sequence to appear, and then looking backwards in time to see if the earlier indicators appeared in the correct sequence and within the allowed time windows.

order within the factor graph: "Look for C, then see if B has occurred in the past, and then see if A has occurred before B." This setup leads to a mismatch between a user's mental model of the sequence and the implementation of the sequence within the code (see Figure 6.4), so we rely on the TEDDI Generation Tool to translate from one to the other.

When traversing the factor graph, TEDDI uses the following procedure to determine the presence of sequences:

- If the node represents an indicator at the end of a sequence (i.e., it's the first node TEDDI sees in the graph sequence), TEDDI looks to see if the indicator is currently present. If it is, TEDDI moves to the next node in the current sequence; otherwise, TEDDI moves on to the next sequence in the graph.

- If the node represents an indicator that is not at the end of a sequence, TEDDI examines the last five time periods that this indicator was present. A time period p for an indicator i is defined by the pair (a, d), where a is the timestamp representing when i switched from absent to present (i.e., i was absent at time $a-1$, but present at time a), and d is the timestamp representing when the first time i became absent after a (i.e., $d = a + k$, where i was found to be present from times a through $a + (k - 1)$

but absent at time $a + k$).

If one of these periods occurred both before the last indicator that TEDDI checked and within the allowable time window as defined by the user, TEDDI moves to the next node in the current sequence (or, if this is the first node in the sequence, declares that event to be present). Otherwise, TEDDI moves on to the next sequence in the graph.

- If TEDDI goes through all of the sequences in the graph without finding any of them, it reaches the end of the graph and declares that no events are currently happening.

As an example, consider the following scenario:

- An edge device is installed in a locked cabinet on a utility pole, but the grid operator is concerned about someone cutting through the cabinet wall with a torch and drilling through the potting material surrounding the memory holding the encryption keys for the device, allowing the attacker to then spoof traffic as that device.
- The operator decides to install a light and temperature sensor with the edge device, and defines the following event sequence for his factor graph:
  - *High temperature* caused by the torch.
  - *Light* as the box is penetrated and the attacker searches for the device and memory they want to access. The time window between the high temperature and light indicators is set to ten minutes.
  - Finally, a *tamper signal* from the potting material[1] indicating that someone is trying to remove it. The time window between this indicator and the light indicator is set to five minutes.

[1] Generally, potting materials inside tamper-resistant devices have sensors embedded within them to detect when they are being penetrated.

- When a malicious actor performs the above attack, TEDDI notes the presence of the initial indicators as they appear, but does not actively respond until the last signal—in this case, an alert from the potting material sensors—appears. When this indicator is present, TEDDI goes through its factor graph like so:

    1. TEDDI checks to see if the potting material tamper indicator is present (it is in this case).
    2. TEDDI moves on to the next indicator (light), and examines the last five time periods the light indicator was present. If none of these periods appear within the five minutes preceding the potting material indicator, TEDDI moves on to the next sequence in the graph.
    3. If one of the above light indicator time periods *does* overlap with the five-minute time window, TEDDI moves to the first indicator in the sequence (high temperature) and repeats the checking sequence. If any of the last five high temperature time periods falls within the ten minute window preceding the light period selected above,[2] TEDDI declares that the attack event is occurring, and moves to take the appropriate response.

Note that while the above factor graph traversals are very linear, TEDDI could be adapted to process the sequences in parallel; however, this would require breaking the graph-checking code into individual threads for each sequence, and single-processor edge devices would render this change useless.

6.4 Tamper Information Points (TIPs)

Tamper information points are the eyes and ears of TEDDI, and are responsible for collecting the sensor data needed for decision making. A TIP is assigned to each edge device in

[2]Choosing which of the light indicators to base our window on introduces a conflict between accuracy and performance; see Section 9.2.1 for further discussion of this problem.

the network, and lives either on or near (i.e., within the same cabinet) the device to monitor the surrounding environment. (Exactly where the TIP lives may be dictated by the sorts of indicators you are looking for—for example, if "network disconnect" is in the indicator set, the TIP will need to be on an auxiliary device (such as the SEL 3622 [112]) to avoid getting disconnected along with the edge device.)

Every few seconds, the TIP performs the following tasks:

- First, it takes a snapshot of its monitor values and determines whether any indicators are currently present by comparing these values to operator-defined thresholds. These thresholds are just single numbers in our prototype, so the TIP may need to pre-process the monitor data to fit its threshold setup:
  - The TIP could use a simple heuristic to calculate the value to compare with the threshold. For example, an accelerometer that detects motion in three directions would need its components combined into a single magnitude value for comparison. This setup, however, may not be fine-grained enough to differentiate different event behaviors.
  - The TIP could use a machine-learning algorithm for pre-processing, and output a specific indicator value based on its findings (i.e. it could output 1 for a minor benign shake, 2 for a stronger force possibly generated by an attacker, 3 for a severe shake caused by a natural disaster, etc.). This decision, however, may conflict with our goal of reducing the configuration burden on grid defenders, since they will need to gather the data needed to train the algorithm.)

  When a monitor exceeds its threshold, the corresponding indicator is considered *present* in our factor graph; otherwise, the indicator is *absent*. This set could potentially be expanded to include non-binary indicators, but we can model these cases using our existing binary indicators—for example, "is the current temperature above,

below, or within the normal operating range?" can be captured through the indicators "are we below X degrees?" (too cold) and "are we above Y degrees?" (too hot).

- As indicators appear and disappear, the TIP updates its history counters to save the last five times each indicator was seen. (We chose five to balance the threat of attackers circumventing sequences by repeating indicators with the additional complexity introduced by having these indicator options (see Section 9.2.1 for more details).

- Once the indicator set is built, the TIP sends it to the tamper decision point that manages the TIP. The TDP uses this data for its own decision-making process, and takes the data as a sign that the TIP is still active.

  Along with its data, the TIP sends both its ID and the current timestamp, which the TDP uses to authenticate the message and make sure that the data they receive is fresh. To preserve the integrity of the message, the TIP takes its message, as well as a 64-bit secret key, and generates a hash-based message authentication code (HMAC) [67] with the SHA-256 cryptographic hash function. The MAC is appended to the message, and the TDP verifies this value upon receipt to validate the message's integrity.[3]

- Finally, the TIP runs its indicator set and history counters through its factor graph to make a local tamper decision by itself, and if successful, sends this decision directly to the appropriate TEPs. However, as mentioned in Section 6.3, TIPs only have a *limited* factor graph, and thus may not be able to definitively resolve sequences with regional indicators. When this happens, the TIP defers to its TDP and sends an alert message to ask the TDP for assistance.

[3] A similar authentication scheme is used for TIP-to-TEP and TDP-to-TEP communication.

6.5 Tamper Decision Points (TDPs)

Tamper decision points serve as the final word regarding what tamper events are occurring on a SCADA network. Ideally, TDPs live in centrally-located, higher-security areas of a utility's SCADA network, such as within a substation, but any node in the network is eligible to host a TDP provided it is not already hosting a TIP. (We restricted TDPs to non-TIP nodes because we assume that the edge devices that are hosting TIPs have limited resources, and thus are not suited to host both a TIP and TDP. However, a TIP could be installed to protect the device hosting the TDP from local attacks.) We discuss the optimal placement of TDPs in Section 7.

Each TDP is given a full copy of the system's factor graph, which allows them to make definitive tamper decisions based on the data they receive. Its primary source of data is the set of TIPs it serves, but it can also query external databases (for example, weather feeds or utility incident databases) to make tamper decisions. (We added a MySQL connection to our original TDP prototype to demonstrate the feasibility of the idea, but given the wide range of potential data sources, we did not include external data sources within the TEDDI Generation Tool.) While every TIP in the system is paired with a specific TDP, the exact number of TIPs that each TDP serves is up to the network administrator: More TIPs give the TDP a better sense of the network's overall tamper state, but servicing too many decision requests may overwhelm the device the TDP lives on.

Each TDP is made up of three threads that are spawned when the program starts:

Heartbeat Thread: This thread is responsible for processing the non-alert messages coming from its TIPs. Upon receiving a message, the heartbeat thread verifies that the message is fresh and came from a legitimate TIP, and if so, it stores the set of local indicators from the message and updates its history counters accordingly.

The heartbeat thread handles regional indicator calculations using a simple majority-

rules voting scheme that involves the corresponding local indicators. For example, if a TDP needs to know the value of a regional shaking indicator, it takes the local shaking indicator values from all of its TIPs, and checks to see if a majority of the TIPs are currently experiencing shaking. In general, the TDP constructs its set of regional indicators $I_{regional}$ as follows:

$$I_{regional} = \{maj(i_1), maj(i_2), \ldots, maj(i_{|I|})\} \tag{6.1}$$

$$maj(i_p) = \begin{cases} 1 & \text{if the majority of TIPs} \\ & \text{see } i_p \text{ as } \textit{present} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

If any of the $maj(i_p)$ functions do not find a clear majority (i.e. the TIPs are evenly split on sensor p), the TDP has several options:

- It can break ties arbitrarily, and declare that an even split will always be reported as present or absent.
- It can reach out to a *peer TDP* for assistance. The requesting TDP sends all of its state data to its peer, including current indicator values, past indicator histories, and other important bookkeeping information such as the number of TIPs that the requesting TDP serves. The peer TDP can then combine its own TIP data with that of the requesting TDP, creating a new joint set of regional indicators to use in making an event decision.

  Using the combined regional indicator set, the peer TDP can make an event decision and pass it back to the requester, who can then pass it on to the proper enforcement points as needed. (While this functionality was included in our original TEDDI prototype, we did not include it in the TEDDI Generation Tool.)

Once the current regional indicators are calculated, we update their history counters in the same manner as the local indicators. (Note that the heartbeat thread does not deal with external indicators; these indicators are processed by the alert thread.)

Alert Thread: This thread is responsible for handling decision assistance requests from TIPs. Upon receiving an alert and verifying that it comes from a legitimate source, the TDP takes the following steps:

- First, the thread waits a short period of time to receive messages from other TIPs. This is done to make sure we have current data from all of the TIPs before we try to make an event decision—otherwise, if a global indicator was present, the system would not recognize this fact until half of the TIPs had reported it, and might make an incorrect decision for early-responding TIPs.

  In our evaluations, our initial waiting time of 1 second proved to be too long, as it causes the TDP's alert socket queue to overflow in the face of steady traffic (while also potentially providing the attacker more time to act before a response could be executed). However, reducing this time too much opened TEDDI up to TIP synchronization issues: If some of a TDP's TIPs were slower in providing fresh data to the TDP due to clock drift, a TDP may be forced to make decisions with stale, inaccurate data. We eventually settled on a wait time of 100 milliseconds to balance these issues, but this duration could be adjusted by a grid defender if necessary.

- Next, the TDP determines the status of its external indicators by querying the data sources associated with the indicators. For example, if TEDDI requires data from a SQL-based incident database about a potential maintenance visit, it queries the database to see if the current time falls within a known maintenance window for the edge device in question.

- Once all of the external indicators are collected, the TIP combines them with

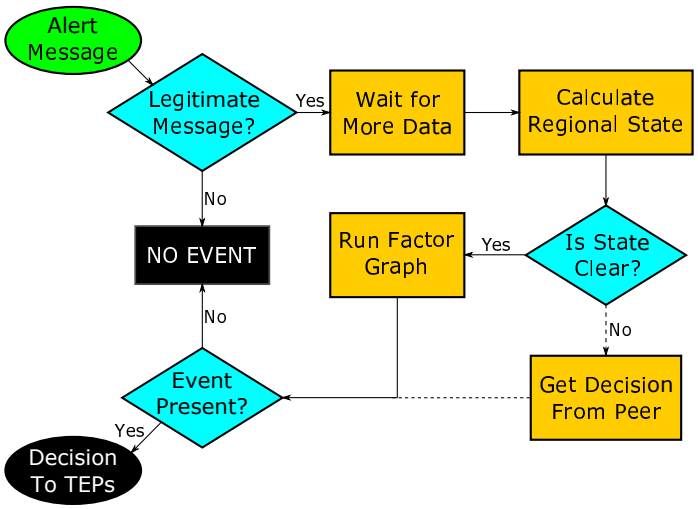


Figure 6.5: A flowchart depicting how a TDP's alert thread responds to a message. The dashed lines indicate that getting help from a peer TDP is an optional step.

both its regional indicators and the local indicators of the TIP who sent the original alert. The TDP runs the entire set through its full version of the factor graph, which allows the TDP to make a definitive, fully-informed event decision.

- Once an event decision is made, the TDP sends its decision to the appropriate TEPs to handle the system's response.

A flowchart of this process can be found in Figure 6.5.

Audit Thread: This thread runs periodically to look for "lost" TIPs and remove them from future calculations. TIPs are considered lost if they do not report data within a specified timeout window (set to 60 seconds in our prototype).

If a TIP is lost, a "lost TIP" event is sent to the appropriate TEPs to further action.

While TDP peering was left out of the current generation tool, for the sake of completeness we define a fourth thread for handling peer assistance requests:

Helper Thread: This thread is responsible for servicing assistance requests sent by other TDPs. When it receives a request, which includes the local data from the TIP who initially sent the alert, the helper thread takes the following steps:

- It combines the TIP data from the requester TDP with its own TIP data, and uses that to create an indicator set $I_{combined}$ that represents the regional indicator state of both TDPs. (If the state is still unclear, the helper thread could request help from another peer TDP, and the pattern could continue arbitrarily until either a TDP finally has enough data to get a clear idea of the regional state, or we run out of TDPs to query. This feature, however, was not implemented in our prototype.)
- It takes both the indicators I_{local} from the TIP who initially sent the data, fuses it with $I_{combined}$, and runs the whole set through its full factor graph. Once a decision is reached, it sends the decision back to the requesting TDP.

Finally, once the alert thread decides that an event is present, it sends its decision to the appropriate tamper enforcement points to handle the response.

6.6 Tamper Enforcement Points (TEPs)

Tamper enforcement points are positioned between the TIP and its TDP on the SCADA network, and they are responsible for responding to decisions made by these devices.

For each TIP, an *edge TEP* is installed at the TIP's location, and a *central TEP* is placed closer to the TIP's associated TDP. The two TEPs allow us to execute responses in the location that makes the most sense in the current context—for example, erasing secret data on the edge device is best handled by the edge TEP, while filtering network traffic might be more appropriate for the central TEP.

Unlike most tamper response systems, in TEDDI TEPs are not limited to making a single response to a decision, but instead can execute an ordered series of responses to mitigate any problems. Responses themselves are defined by three attributes:

- The shell script (provided by the user) that defines the actions that make up the response.
- The severity classification of the response. Responses can be classified as either weak, moderate, or strong.
- Whether or not the response is repeatable.

These attributes allow for flexibility in TEDDI's response, which is critical when addressing the grid defender's dilemma, given the costs associated with taking the wrong response to an event (Section 4).

The system operators set exactly which responses to take for each event, as well as what order to take them in, using the TEDDI Generation Tool. In cases where responses need to be taken by both the edge and central TEPs, the TEPs coordinate their response to ensure that responses occur in the proper order.

6.7 Limitations of TEDDI

While the above architecture offers a number of advantages over existing systems, it is not without its drawbacks:

- TEDDI is also vulnerable to "low and slow" attacks; that is, attacks that progress slowly enough that one or more of the sequence's timing windows are exceeded. This technique is commonly used when exfiltrating data from a system, and could be used to great effectiveness here. However, if an attacker must be on location to physically tamper with a device, slowing down their attack may increase their chances of being noticed in other ways (such as being spotted by a passerby).

  This attack has a parallel in the network intrusion protection sphere, where packet fragmenting can lead to similar issues with sequences and timeouts, albeit involv-

ing network traffic rather than environmental indicators [114]. Oftentimes, attacks against a network intrusion protection system (NIPS) exploit differences between the timeout values of the NIPS and the system it is trying to protect—for example, if a NIPS waits for x seconds before discarding fragmented packets, but the protected system waits $y > x$ seconds before doing so, an attacker can send a packet fragment every z seconds such that $x < z < y$, causing the fragments to sneak past the NIPS undetected and still get reassembled on the victim system.

The primary reason TEDDI is vulnerable to slower attacks, therefore, is that the host system has no mechanism for "discarding" physical indicators, meaning that y is essentially infinite in this scenario. That means that patient attackers only have to wait for the x time window to expire before continuing their attack. (However, since x is different for each time window, the attacker may have trouble determining how long they have to wait.)

- Because TEDDI only reports one tamper event at a time (the highest-ranked one that it sees), improperly ranking events could lead to two problems:
  - If a high-priority event is ranked too low, then TEDDI may miss the event if it appears at the same time as a higher-ranked (but actually lower-priority) event.
  - If the sequence for an event E_i is a exact subset (i.e., it has same indicators, in the same order, and the same time windows) of a sequence for the event E_j, then if E_i is ranked ahead of E_j, TEDDI will never detect E_j because it will always look for (and find) E_i first.

These issues make it possible for an attacker to mask an event by inducing a concurrent sequence that corresponds to a higher-ranking event. However, the attacker must be careful when taking this approach, as the response to the higher-ranking event may still interfere with the event the attacker wants to mask. Additionally, TEDDI could be modified to report any number of events that happen simultaneously, rather than

just the event with the highest rank; however, this change may complicate our response mechanisms, as the actions taken for one event may be the very actions we want to avoid for another.

- While we assume that the SCADA network is reliable and always available, this may not always be the case in the real world. TIPs can still perform local event detection when isolated, but if a regional indicator is needed, the TIP simply sends an alert to the TDP and assumes that the problem will be take care of by someone else, which could lead to problems if the TDP is not available.

- A single component failure in our system may leave the systems supported by that component open to compromise. For example:

  - If a TIP fails, TEDDI will no longer receive indicator data from the corresponding edge device. Eventually, the TDP will recognize that the TIP is no longer active, and stop including its data in the TDP's regional state calculations.

  - If one of the threads spawned by a TDP fails, all of the TIPs managed by that TDP are put at risk. Without the heartbeat thread, for example, the TDP would not be able to collect current indicator data from TIPs, and would have to rely on stale data to make event decisions (and eventually lose its ability to calculate regional indicators). Losing the alert thread, on the other hand, would keep TEDDI from responding to any event with a regional indicator, as the TIPs trust that the TDP will always be there to assist them and will blindly send alerts even when the TDP disappears.

  - Losing a TEP will effectively turn TEDDI into a detection-only protection system for the corresponding edge device, as the TDP and TIP will continue sending event decisions with the assumption that the TEP is available to handle the response. This issue may arise even if only one of a TIP's TEPs (either edge or

central) is lost, as the remaining TEP will likely end up waiting for the lost TEP while trying to coordinate its response.

- The indicators available to TEDDI are constrained by the monitors available to the operator, which means that there may be an added cost to deploying TEDDI in the form of installing monitors around edge devices. (However, some manufacturers have started building monitors into their products—for example, the SEL 3622 features an accelerometer and light sensor [112].)

- Finally, the most glaring issue with TEDDI is that as a signature-based protection system, it is only as good as the signatures (or sequences, in this case) that it has. A grid defender may have an incorrect mental model of a tamper event, and end up assigning an incorrect indicator sequence to that event, causing TEDDI to miss the event when it eventually occurs. Some possible pitfalls include:

  Excluding or misordering important indicators in a sequence. Omitting indicators is not as big a problem as it might seem, since the event should still match the indicators that are included in the graph (provided the time windows are long enough). For example, if an event E is made up of the indicator sequence ABC, the event will still match the sequences AB, AC, or BC if one of the indicators is left out of the graph.

  Misordering indicators, however, is a bigger problem, as this will cause TEDDI to miss the event completely: While looking for AB will still catch the sequence ABC, looking for ACB will not.

  Adding extraneous indicators to a sequence. An operator may over-think the process and add unnecessary indicators to a sequence.

  Choosing an improper response set for an event. Even if an operator defines a suitable sequence for an event, they may still select an incorrect set of responses to apply.

One possibility for mitigating this problem is to add an early-response mechanism that allows grid defenders to apply a subset of the event's responses when we see the initial portion of the event sequence. This is part of the motivation behind the "pre-event" portion of our Response Suggestion Engine, which we discuss in Section 7.2.

Another issue with our signature structure is that multiple sequences may need to be defined if the exact sequence for an event is ill-defined (for example, "A, then either B or C, and then D"). This may force an operator to define several sequences to capture all of the potential possibilities, increasing the chances that they will make one of the sequence mistakes mentioned earlier.

All of these issues underscore the importance of building an accurate, comprehensive factor graph containing the events a grid defender is worried about.

The challenge of avoiding the above pitfalls while building a factor graph, combined with the fact that a unique program must be generated for every TIP, TDP, and TEP a network requires, suggests that building a TEDDI system, and in particular putting together a proper factor graph, may be a challenge for grid defenders, and that constructing a tool that helps automate this process would be very useful. For example, we could suggest an early response if the beginning indicators of a long sequence are present, or try to fill in gaps in response sequences for events that appear. To address these concerns, we designed the TEDDI Generation Tool, which we discuss in detail in the next chapter.

Chapter 7

The TEDDI Generation Tool

In this chapter, we describe the *TEDDI Generation Tool*, a program that takes information about the network, the devices, and the events we want to detect, and outputs the necessary custom programs to deploy TEDDI on any arbitrary network.

The tool itself is written in a combination of PHP, C, and MySQL, and encompasses over 10,000 lines of code in total. The tool has two major components: A TEDDI website that collects the data from the user and translates it into intermediate files representing TIPs, TDP, TEPs, and the full factor graph; and a code generation program that turns the intermediate files into the C files that then be compiled and deployed on their destination hosts.

To show how the generation tool works, consider the following example: A grid defender is deploying a set of edge devices around the utility's service area, and is looking to build a TEDDI system to protect them. Upon accessing the TEDDI website, the defender goes through the following process:

1. The user enters the events, indicators, and monitors that will make up TEDDI's factor graph. For example, if our operator is concerned about someone breaking into an edge device cabinet and plugging a malicious USB drive into the edge device (as

described in Chapter 3), they could define the following items:

- Event: “Malicious USB Drive.”
- Monitors: A cover switch, a photosensor, and a USB drive cap (to keep the port covered). We also include a link to query an external database for an extra indicator, which we explain below.
- Indicators: “Open Door,” “Presence of Light,” “USB Cap Removed,” and “USB Accessed.” However, if firmware updates or other patches can be provided via USB, a fourth indicator “Unscheduled Activity” can be added to interface with a utility’s incident database and see whether the device is scheduled to be serviced. This fourth indicator will also be marked as accessible only by the TDP.

2. Next, the user links the monitors and sensors, and then defines the indicator sequences that make up the events:

   - The Open Door indicator is linked to the cover switch, the Presence of Light indicator is linked to the photosensor, USB Accessed links to the drive cap, and Planned Maintenance is joined to our external database. Since the cover switch and drive cap are binary sensors that are either Open or Closed, their thresholds are simply set to Open, while the photosensor threshold is set to ensure it is above the amount of light emitted by the device’s own LEDs. The database threshold is set depending on the query that is made—for example, if we just ask if an incident ticket is currently open for this device, the threshold can be set as binary (there is a ticket, there is no ticket).
   - The indicator sequence order for the Malicious USB Drive event is defined as “Open Door, then Presence of Light, then USB Cap Removed, then USB Accessed, and finally Unscheduled Activity.” The time window between the

first two indicators is set relatively short (i.e. a few seconds), as either ambient light will immediately strike the the device, or the attacker will have a light source of their own that they will shine on the device. The time window between light and the USB device will only be slightly longer, on the order of a few minutes, as we assume the attacker will not want to wait around once they have the cabinet open and will thus proceed quickly with their attack. (For the external database, the TEDDI-defined time window is ignored in favor of the window defined by the incident ticket, i.e. "A technician will be at this box between 10 AM and noon on April 21st.")

3. Now, the user enters the responses available to the system, and links them to the events. For our USB event, the utility may want to send an alert back to their control center (which they label as a Weak response), begin monitoring traffic for suspicious behavior (a Moderate response), and disable the port to keep the attacker off their device (which could be labeled as either Moderate or Strong).

4. Next, the user defines the topology of their SCADA network, and decide where the TIPs, TDPs, and TEPs should be placed. In our example, the operator uses the Network Topology Uploader (see Section 7.3) to quickly inform TEDDI of their network layout and tell TEDDI that their USB-vulnerable device (and every other device like it) requires a TIP for protection. The TDP Placement Tool (Section 7.4) is then called upon to place TDPs, and determine which TDP should be linked to our device.

5. Finally, the user enters the IP/port information for all of the devices on which TIPs, TDPs, and TEPs will live. Once done, the generation tool spits out the TIP code then can then be compiled and deployed on the edge devices.

The generation tool includes four important components that assist the user in constructing their TEDDI system: the Factor Graph Domain-Specific Language, the Response

Suggestion Engine, the Network Topology Uploader, and the TDP Placement Tool. We now describe each of these components in detail.

7.1 Factor Graph Domain-Specific Language (FGDSL)

To accomplish our goal of capturing an operator's intuition about the events they want to detect, the generation tool provides a *domain-specific language (DSL)* [124] based on our factor graph, which we call *FGDSL* (factor graph domain-specific language). DSLs have been used to simplify and streamline a diverse set of tasks, ranging from generating video card drivers [130] to creating and verifying cache coherence protocols [24] to streamlining software production for the Estonian customs system [41]. FGDSL provides similar benefits by letting operators define components and component relationships at a high level, and then automatically translating them down into C code.

FGDSL consists of four basic data structures: Events, indicators, monitors, and responses. The structures are formally defined in Figure 7.1.

On top of the basic structures, we define three important relations:

- A one-to-one *indicator/monitor* relation, which links indicators to the monitors that look for them.[1] The threshold for this relation is defined here as well.
- An ordered, one-to-many *event/indicator* relation, which links events to the indicator sequences that define them. This is also where the allowed time windows between indicators are defined.
- And ordered, one-to-many *event/response* relation, which links events to their respective response sequences.

[1] We define indicator/monitor relations as one-to-one to simplify our monitor-reading code, but could expand our prototype to link a monitor to several indicators if desired.

```
Event
{
        char[50] name
        int rank
}
```

(a)

```
Indicator
{
        char[50] name
        bool startSetting
        int level
}
```

(b)

```
Monitor
{
        char[50] name
        char[100] location
        float initValue
}
```

(c)

```
Response
{
        char[25] name
        int class
        int strength
        bool isRepeatable
        char[150] script
}
```

(d)

Figure 7.1: The definitions of (a) events, (b) indicators, (c) monitors, and (d) responses in FGDSL.

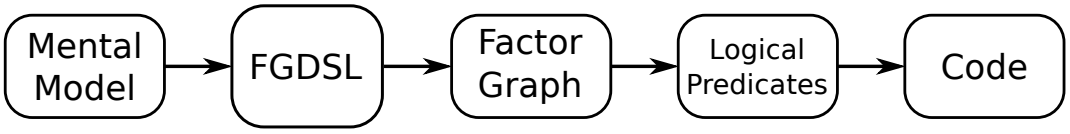


Figure 7.2: A diagram of the model-to-code conversion process in TEDDI. Note that the user only needs to complete the first transition to FGDSL; everything else is managed by the TEDDI Generation Tool.

With these primitives, a grid defender can easily translate their mental model of an event directly into FGDSL. From here, our generation tool can build a factor graph, define the logical predicates that make up the factor functions of the graph, and eventually generate the low-level C code that looks for the events. The conversion process is shown in Figure 7.2, and described in more detail in Section 9.4.

7.2 Response Suggestion Engine

The assigning of responses to events is a manual process, and can be a little tedious even with the Generation Tool. Therefore, we included the Response Suggestion Engine to help

users determine response strategies for events with long or similar indicator sequences. After the user completes a first pass of linking responses to events, the engine does the following:

Pre-Event Suggestions: The suggestion engine has two sequence length thresholds a and b, which by default are set to four and six, respectively. (These numbers are based on our assumption that indicator sequences will generally contain fewer than ten indicators, but could be tuned at a later point if necessary.)

If e's indicator sequence i falls in the range $a < |i| < b$, and e's response sequence begins with one or more weak responses, the system suggests defining a new "pre-event" e^* in the factor graph, allowing the user to take some pre-emptive steps for a larger event that may be coming. e^* is given the following attributes:

- e^* will be ranked behind all of the other events, to make sure we do not overlook a full event in favor of a pre-event.
- The indicator sequence for e^* will be set as the first half of e's sequence (which is two by default).
- The response set for e^* will be the weak responses that begin e's response set, and they will be ordered the same as in e. For example, if ABC is the ordered set of responses we take for e, A is classified as a weak response, and B is classified as a moderate response, the response set/order for e^* will just be A.

If $|i| < b$, the system suggests defining two pre-events e^* and e^{**} in the factor graph. e^* is defined exactly as described above, but e^{**} is constructed as follows:

- e^{**} will be ranked behind all other events, but ahead of e^*.
- The indicator sequence for e^{**} will be set as the first *four* indicators in e's sequence.

- The response set for e^{**} will be the weak *and moderate* responses that begin e's response set, and they will be ordered the same as in e. For example if we consider the same response sequence ABC from above, the response set/order for e^{**} will be AB (and potentially ABC, if C is not a strong response).

For example, suppose we are considering the Schweitzer Scenario from Section 3.4 as one of five events in our factor graph, and we have determined that our response will be to alert the control center (a weak response) and then sever the edge device's network connection (a strong response). When we examine the problem, we wind up extracting four indicators: *Shaking*, *cover open*, *light present*, and *network disconnected* (see Section 8.2 for more details on this breakdown). This sequence would cause TEDDI to suggest the following pre-event e^*:

- We set e^*'s rank to six, to ensure that we consider all of the full events before we start looking for pre-events.
- e^*'s indicator sequence will consist of the first two indicators from the Schweitzer Scenario—in this case, *shaking* and *cover open*.
- e^*'s response sequence will just include alerting the control center, as the other response (severing the network) is considered too strong for a pre-event.

Pre-events assist grid defenders in two ways:

- They reduce TEDDI's reaction time to an event by letting TEDDI take pre-emptive steps towards mitigating an event before the event actually occurs.
- They help guard against incorrect event sequences by only considering the start of the sequence. For example, if a defender sets e's indicator sequence as $ABCD$, but e's true sequence is $ABCE$, TEDDI will suggest a pre-event AB that will still be detected even if the full sequence is not.

Response Suggestions: If two events have response sequences that begin the same way, the suggestion engine will offer to combine the sequences such that the same responses are used for both events. More formally, if events e_1 and e_2 have corresponding response sequences r_1 and r_2, and there exists a response sequence s such that $length(s) \geq 2$ and $r_1 = sr'$ and $r_2 = sr''$, TEDDI will set both response sequences to st, where $t = (r' \cup r'')$. (In merging the response sequences, however, the tool does not attempt to order the added responses; this task is left to the user once the merging is complete.)

Combining responses for similar events help guard against response omissions—for example, if we consider the Maintenance Mode Attack from chapter 3, if the defender chooses to log the benign event but not the malicious one, TEDDI will ask about adding the log response to the latter event. However, the grid defender needs to carefully review these suggestions before accepting them, as the responses for one event may not be suitable for the other given the difference in event severity.

7.3 Network Topology Uploader

Unlike factor graphs, we assume that grid defenders already have a diagram of their network topology on hand. Rather than forcing defenders to enter the details of their network manually into TEDDI, we want to let them upload their existing topology to the system, and allow TEDDI to use the provided information to fill in the details automatically. This goal led us to construct our network topology uploader.

Network topologies can be expressed in a number of different ways—for example, one industry representative we spoke with stored their topology records in an Excel spreadsheet. TEDDI's topology uploader, however, is based upon Weaver et al.'s Cyber-Physical Topology Language (CPTL) [149]. CPTL takes a SCADA network, complete with infor-

mation about the various devices and the links between them, and depicts it as a high-level graph, where vertices represent devices and edges represent the network links between them. From there, CPTL defines a set of primitives that combines both cyber and physical information sources to describe the network in question (for example, high-level documentation or low-level IP information), as well as a set of operations that can be run on these primitives. CPTL stores graph data and attributes in a JSON-like syntax, which can be seen in Figure 7.3.

CPTL was originally designed as a way to streamline NERC CIP audits,[2] which are estimated to "consume 30 man-days of work per day per audit" [149]. To assist grid defenders in preparing for and demonstrating NERC CIP compliance, the authors define a set of "*vertex and edge attributes*" [149] that augment the graph with domain-specific information (device classifications, IP information, etc.), as well as a pair of operations to support network expansion and contraction. These operation help grid defenders visualize their networks by allowing them to quickly view information at the desired level of granularity, thereby making it easy to evaluate the vulnerability and configuration status of their devices.

For TEDDI, we lean on CPTL's vertex and edge attributes to determine the structure of the given network. More specifically, we look for specific information within the JSON-based storage files that give us clues as to where TIP and TDPs should be placed:

- CPTL's *rdfs:type* vertex attribute tells TEDDI what sort of device this node is, and whether or not it needs to be protected. Types such as "Generator," "Meter," or "Recloser Control" indicate that we should place a TIP on or with these devices, while types such as "Node" or "Meter Controller" indicate potential TDP locations. In Figure 7.3, for example, the `syard:Generator` attribute dictates that `Generator1`

[2]NERC stands for "North American Electric Reliability Corporation," and their Critical Infrastructure Protection (CIP) standards are mandatory rules regarding cybersecurity controls, documentation, and reporting/recovery processes needed by covered utilities [84]. Failure to adhere to these standards can lead to stiff penalties, such as fines as high as $1 million per day that the standard was not followed [149].

```
{
   "nodes":
   [
              {"name":"substation-yard:Generator1",
               "rdfs:type":"syard:Generator"},
              {"name":"substation-yard:Node1",
               "rdfs:type":"syard:Node"}
    ],
    "links":
    [
              {"source":"substation-yard:Generator1",
               "target":"substation-yard:Node1",
              "relation":"syard:hasLine"}
    ]
}
```

Figure 7.3: An example of a network topology file used by CPTL [149] and TEDDI. TEDDI uses the `rdfs:type` field of nodes to decide where TIPs and TDPs should be placed, and uses the `relation` field of links to see which nodes are directly connected.

will require a TIP for protection, while the `syard:Node` attribute of Node1 signals that we should consider hosting a TDP on this device.

- CPTL's *relation* edge attribute—more specifically, the *hasLine* relation value—tells TEDDI which devices are directly connected. (In Figure 7.3, the `hasLine` shows there is a communication path between `Generator1` and `Node1`.) We can deduce the general topology of the network through this relation, and it is also vitally important to the TDP Placement Tool (Sec. 7.4).

The CPTL syntax also offer the potential for expanding the topology uploader and further reducing the burden on TEDDI's users. For example, the "CPTL enterprise namespace... incorporates more detailed information about devices on a network via vertex attributes such as IP address" [149], which could allow users to avoid manually entering IP information for TIPs, TDPs, and TEPs.

Once a SCADA network is uploaded, placing TIPs and TEPs is a straightforward process: Every device in the network that we want to protect requires a TIP to be installed with

it, and TEPs default to the locations of the TIPs (edge TEPs) and TDPs (central TEPs). Placing TDPs is a trickier process, but a process that is made much easier by our TDP Placement Tool.

7.4 TDP Placement Tool

The TDP Placement Tool assists a grid defender in placing tamper decision points within their network. Any non-TIP device in the SCADA network is a potential location for a decision point, but we want to *minimize* both the number of TDPs we have to place in the network and the distances between TDPs and their TIPs, in order to simplify the installation process for the utility.

Thankfully, our question generalizes to a well-studied problem within computer science: the *simple plant location problem (SPLP)* [52]. In the general form of this problem (known as the *fixed cost median problem (FCMP)*), we wish to find the optimal locations to build service centers such that we minimize our total costs, which consist of i) the cost of building these centers, and ii) the cost of servicing other locations in the area from the centers they build. The SPLP represents the discrete median case of the FCMP, which seeks to solve the following problem: Given the set of locations L we want to serve, the set of potential plant locations P, d_{lp} is the cost of serving location $l \in L$ from location $p \in P$, and f_p is the cost of placing a facility at p, find the optimal subset $S \in P$ that minimizes the cost $Z(S)$. Defined formally by Hochbaum [52] (note that we have changed a few of the variable names from his equation):

$$\min_{S \subset P} Z(S), \text{ where } Z(S) = \sum_{l \in L} \min_{p \in S} d_{lp} + \sum_{p \in S} f_p \tag{7.1}$$

In our case, our set L is the set of TIPs in the networks, whereas P denotes the potential TDP locations, d_{lp} is the number of hops between l and p in the graph (for now, we weight

all links equally and do not consider how they translate into actual distance), and f_p is 1 for all $p \in P$. With these definitions, solving the simple plant location problem will provide a solution that serves all of our TIPs with the fewest number of TDPs.

Unfortunately, finding an optimal solution to the SPLP is not easy, as we can reduce a known NP-complete problem (Set-Cover) to the SPLP, thus showing that the SPLP is also NP-complete [52]. However, we use a greedy Set-Cover algorithm to produce an initial TDP set that approximates the optimal solution [30].[3]

We define our Set-Cover algorithm as follows:

- We start with the formal definition of the problem: "An instance $(X, \mathcal{F})$ of the set-covering problem consists of a finite set X and family $\mathcal{F}$ of subsets, such that every element in X belongs to at least one subset of $\mathcal{F}$...The problem is to find a minimum-sized subset $\mathcal{C} \subseteq \mathcal{F}$ whose members cover all of X" [30].
- We define X as the set of TIP-protected nodes in our graph that are *adjacent* to a potential TDP node. (We deal with TIP nodes that are only adjacent to other TIP nodes in a separate step after we run our algorithm.)
- For each potential TDP node t in our graph, we define a set f_t that contains all of the nodes in X that are adjacent to t.

When the greedy Set-Cover algorithm is run with these definitions, it will choose the node adjacent to the most TIPs in every round. Therefore, our solution set $\mathcal{C}$ will contain the nodes on which we should install TDPs, and tell us which TIPs each TDP should service. With this approach, we obtain a near-optimal solution for both the number and position of our decision points.

While any node that is not a TIP is considered a potential TDP landing spot, in practice there may be nodes that are not suitable for hosting decision points. This issue can be

[3]Specifically, the set produces by Greedy Set-Cover is $H(d)$ time larger than the optimal, where $H(d)$ is the dth harmonic number [30].

solved in one of two ways: either the Network Topology Uploader can be tweaked to account for non-TIP, non-TDP nodes, or the user can adjust the TDP layout after-the-fact by re-assigning TIPs to other decision points. (The former method is preferable, as the latter forces defenders to manually tweak their TIPs every time they create or update a TEDDI system.)

While greedy Set-Cover will satisfy most of our needs, a minor problem remains: If a TIP is not immediately adjacent to a potential TDP node, it will get left out of our algorithm and not get connected to a TDP. To address this issue, once we have a TDP set from greedy Set-Cover, we run a simple breadth-first search (BFS) from each TIP node that was not included in the algorithm, and connect it to the first TDP it finds.

In terms of the its runtime, the algorithm breaks down as follows for a network featuring the set D of TDP-eligible devices, the set A of TIP-protected devices that are adjacent to nodes in D, the set N of TIP-protected devices that are *not* adjacent to nodes in D, and the set K of network links:

1. Our initial splitting of the nodes into TIP and TDP groups runs in $\Theta(D + A + N)$ time.

2. Constructing the subsets for our Set-Cover algorithm requires $O(D(D + A))$ time.

3. Identifying and setting aside the excluded nodes takes $O(K(A + N))$ time.[4]

4. Our Set-Cover implementation gives us a runtime of $O(AD(1 + A))$ time.[5] This differs from Cormen et al.'s runtime of $O(|A| * |D| * \min(|A|, |D|))$ [30] due to a slight difference in implementation.

5. Our BFS implementation matches Cormen et al. more closely, and gives us an equivalent runtime of $\Theta(N) * O(D + A + N + K)$ [30].

[4]This could also be written as $O((A + N)(D + A + N))$ time, i.e. "we might have to check every node as a potential neighbor" versus "we might have to traverse every network link to find the neighbors."

[5]This could also be written as $O(D^2(1 + A))$, as the number of outermost loop iterations are upper bounded by both the number of sets (i.e., the number of TDPs) and the number of TIP nodes in A.

The resulting combination runs in quadratic time, but the term that ends up dominating the equation will depend on the relative sizes of D, N, A, and K. (Our performance evaluation, on the other hand, depicts a linear relationship between system performance and the overall number of network nodes; we discuss the potential reasons why in Section 9.3.)

Finally, we note that the BFS term's impact could be minimized by expanding the definition of sets within Set-Cover to include nodes some number h of hops away from a TDP-eligible node, but how this might affect the runtime of greedy Set-Cover is unclear.

7.5 Generation Tool Limitations

The TEDDI Generation Tool helps improve the usability of the system by simplifying the process of creating a TEDDI system, but updating an existing TEDDI system (for example, adding a new event to a factor graph) can be a bit of a hassle because the system has to be re-generated and re-deployed to all of the devices in the system. For example, while TIPs will automatically reconnect to a TDP that is taken down and updated, a TIP that disappears for too long will eventually be considered "lost" by the TDP, and the TIP will no longer be able to send messages or alerts to TEDDI. These sorts of issues, however, could be addressed by making the system a bit more modular (for example, a TIP could read in a factor graph file upon startup) such that a TIP or TDP would not have to be stopped to be updated.

Chapter 8

TEDDI in Action

In this chapter, we demonstrate the effectiveness of TEDDI by revisiting the tamper scenarios from Chapter 3, and show how TEDDI addresses the issues raised by the grid defender's dilemma.

8.1 Scenario 1: Device Credential Heist

Our first scenario breaks down nicely into a sequence of indicators: First the attacker *opens* (o) the device case, uses a *light source* (l) to locate the protected memory chip, and then attempts to pierce the potted *mesh* (m) and *probe* (p) the chip underneath to extract the secret key. In addition to the device's mesh, then, we need a cover seal/switch, photosensor, and a special probe sensor [151] in our monitor set. (While these would most likely installed specially for our purposes, manufacturers are starting to include monitors is in their products, such as the SEL 3622 [112].)

The time windows for this scenario are set as follows:

- The window between o and l can be fairly short (sixty seconds or less), since ambient light will be let in as soon as the device opens, and the attacker will use an external

light source (for example, a flashlight) if there is not enough ambient light.[1]

- The l-m window can be set to be a bit longer (perhaps several hours), as we expect an attacker to be more cautious as he or she tries to penetrate the mesh's potting material without tripping the mesh sensors.

- Finally, the m-p window can be somewhere in between the prior two (roughly sixty minutes), as the attacker may need some time to place the probes while continuing to avoid the sensor mesh.

(We note here that it can be difficult to validate the exact time windows needed without empirical evidence, and thus grid defenders may want to err on the side of longer windows to ensure that slower attackers are still detected by TEDDI.)

Most of these indicators are binary values (cover open/closed, sensor mesh hit/not hit, probe present/absent), but the photosensor threshold will need to be calibrated to pick up the sharp increase in light that occurs once the device is opened.

For our response sequence, we take the following steps:

- We log the event, both on the device and in the control center.

- We limit the traffic this device is allowed to send through selective revocation of its keys. For example, if the edge device in question is a smart meter, it may still be allowed to communicate with its upstream data aggregator, but not directly back to the utility's control center.

- We attempt to destroy the keys revoked in the previous step, to keep the attacker from using them in the first place.

[1] Attackers striking at midnight who are either equipped with night vision goggles or who know the device well enough that they do not require a light source will require a different indicator sequence.

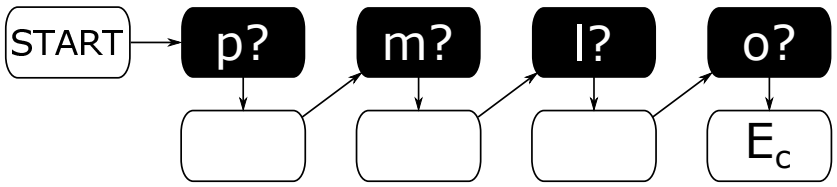


Figure 8.1: A diagram of the factor graph sequence used to represent the Device Credential Heist scenario.

Once the indicators and response sequence is defined, the user's work is finished; TEDDI easily turns the sequence data into a simple factor graph sequence chain for our tamper event E_c (which is shown in Figure 8.1) using the indicators o, l, m, and p.

Next, TEDDI produces a logical representation for our credential heist:

- The event E_c is defined by the indicator sequence $olmp$, where o represents the opening of the device, l represents the presence of light, m represents the the piercing of the sensor mesh, and p represents device probing.
- We define I as the set of k time periods $\{I_1, \ldots, I_k\}$ when the indicator i was present. (In our case, $k = 5$, as we only save and consider the last five periods an indicator is present.) Each period I_j in I has the following attributes:
  - $s(I_j)$: The start time of I_j.
  - $e(I_j)$: The end time of I_j.

  In addition, we use X' to represent a single time period within the set X, and define $w(x, y)$ as the time window for two adjacent indicators x and y.
- We say that E_c occurs if all of the indicators that make up E_c appear in the proper order and within the appropriate time windows.

The final predicate is arranged as follows:

$$
\begin{aligned}
&\exists\, P' \in P \wedge \\
&\exists\, M' \in M \wedge (s(M') \leq s(P') \leq (e(M') + w(m,p))) \wedge \\
&\exists\, L' \in L \wedge (s(L') \leq s(M') \leq (e(L') + w(l,m))) \wedge \\
&\exists\, O' \in O \wedge (s(O') \leq s(L') \leq (e(O') + w(o,l))) \;\Rightarrow E_c
\end{aligned}
\tag{8.1}
$$

In other words, if there exists time periods $P' \in P$, $M' \in M$, $L' \in L$, and $O' \in O$ such that they occur in the proper order and within the proper time windows, then we say that the event E_c is occurring.

Finally, TEDDI translates the graph and its factor function into C code that will be included in the various TIPs and TDPs.

8.2 Scenario 2: The Schweitzer Scenario

The attack scenario that originally inspired TEDDI lines up nicely with the data required by FGDSL. Both the event and the indicators (and the indicator sequence) can be are described straightforwardly: We first look for *shaking* (s), i.e. significant movement generated by the attacker's rough treatment of the lock, followed by an *opening* (o) of the cabinet door, followed by *a light source* (l) reaching the device, and concluded by a *disconnecting* (d) of the device's network cable. Likewise, the necessary monitors follow nicely from the indicators that make up the event.

While the timing windows are not explicitly set within the narrative, we assume that an attacker would go through this attack in quick succession and not wait for long periods between each step, as either they would be concerned about being noticed and reported, or they are confident that they will not be noticed (for example, they are disguised as a technician) and will not want to delay their gratification. Therefore, we can set the timing windows to be relatively short, on the order of a minute or two.

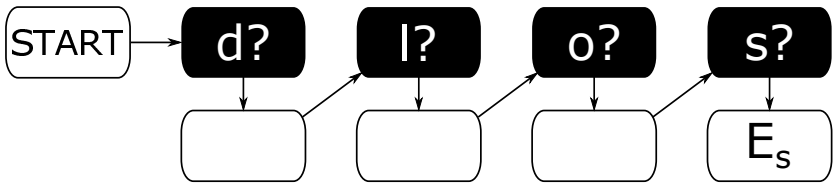


Figure 8.2: A diagram of the factor graph sequence used to represent the Schweitzer Scenario.

Both the response sequence and the monitor thresholds can now be set at the discretion of the operator. If the potential consequences are deemed severe enough, the operator can choose to isolate the compromised box from the rest of the network until someone can be dispatched to fix it. Thresholds can be set relatively low or high (or better yet, measured experimentally) to strike the proper balance between false positive and negatives.

Finally, we must address an important point: Disconnecting the network of the edge device could cause a problem if the TIP lives directly on the device. Therefore, we must place the TIP on a separate device inside the cabinet of the recloser control, and route the edge device's network connection through this separate device. (In fact, Schweitzer anticipated this scenario and placed its sensors in the SEL-3622 Security Gateway [112], which lives with the edge device and provides authenticated access to the SCADA network.)

From here, the user's work is finished, and TEDDI can easily turn the sequence data into a simple factor graph sequence chain (shown in Figure 8.2) using the indicators s, o, l, and d.

Next, TEDDI produces a logical representation of the Schweitzer Scenario event E_s:

- The event E_s is defined by the indicator sequence $sold$, where s, o, l, and d represents shaking, opening, light detection, and a network disconnect, respectively.
- We say that E_s occurs if all of the indicators that make up E_s appear in the proper order and within the appropriate time windows—that is, if there exists time periods $D' \in D$, $L' \in L$, $O' \in O$, and $S' \in S$ such that they occur in the proper order and

within the proper time windows, then we say that the event E_s is occurring.

Using the definitions of I, $s(I_j)$, $e(I_j)$, and $w(x, y)$ from our previous case study, we now build the logical predicates for our events:

$$\begin{aligned} &\exists\, D' \in D \wedge \\ &\exists\, L' \in L \wedge (s(L') \leq s(D') \leq (e(L') + w(l, d))) \wedge \\ &\exists\, O' \in O \wedge (s(O') \leq s(L') \leq (e(O') + w(o, l))) \wedge \\ &\exists\, S' \in S \wedge (s(S') \leq s(O') \leq (e(S') + w(s, o))) \;\Rightarrow E_s \end{aligned} \tag{8.2}$$

Finally, TEDDI translates the graph and its factor function into the C code that will be included in the various TIPs and TDPs. Thus, the event sequence derived from our conversations with SEL maps nicely into our factor graph sequences, and TEDDI can easily and accurately adapt this sequence into its code.

8.3 Summary: Scenarios 1-2

The above two scenarios address the following points of the grid defender's dilemma:

Current protection systems have either no response or a single response. For example, when faced with a similar scenario, the IBM 4758 will "erase its secrets and shut itself down" [119] the moment it detects a tamper attempt. However, if the device was part of a smart grid edge device, such a shutdown could impact the availability of the grid. TEDDI, in contrast, is not locked into a response that is overkill for many situations, and in these cases, we can tailor our response to mitigate the attack while minimizing our impact on grid operations.

Current protection systems are reactionary. TEDDI's sensor setup gives it up an advantage over many other protection systems (especially those in the intrusion detection

sphere) by responding to attacks earlier in their "kill chains" [54]. In the above scenarios, the moment a probe is placed or the device is disconnected, TEDDI detects the event and springs into action, and quickly goes through its factor graph (see Section 9.3), makes a decision and begins executing its response before the attacker gets a change to plug in their own device or use the device's key. In contrast, other systems (for example, PAC [141]) must wait until the attacker accesses the network and starts sending anomalous traffic before it can act. This quick reaction gives TEDDI a huge advantage over its competition, as it buys the defender precious time in the event of an active attack.

Current protection systems require a lot of manual configuration. Many protection schemes impose an extra configuration burden by requiring grid operators to define complex data structures or collect large datasets to make their systems run properly. TEDDI, in contrast, needs only the information defined above: Events, Indicators, Monitors, Responses,[2] and how they are all linked together. This simpler setup allows grid defenders to collect the necessary information about an event straight from its description, and the TEDDI Generation Tool translates the defender's high-level view of the event into a workable protection system, making the construction of a TEDDI system for any arbitrary SCADA network a simple and straightforward process. (We dive into this topic in more detail in Section 9.4.)

8.4 Scenario 3: Maintenance Mode Attack

For this attack, we can break down the sequence like so: The attacker *opens* (o) the device case and *removes* (rem) the tamper seal, adds and connects their malicious hardware addition to the device, and then *replaces* (rep) the tamper seal with a convincing replica.

[2] Ideally, responses have already been defined as part of the utility's incident response protocols, and can be easily worked into the TEPs.

However, because a legitimate technician would perform the same tasks, when servicing the device, we need an additional external indicator to indicate if the service is *scheduled* (s). (One interesting note: We exclude the light indicator because the meters that were tampered this way were covered by a transparent front [33], and so the indicator would not be useful to include in our sequences.)

We set the time windows for the sequence denoting the malicious event E_m as follows:

- We set $w(o, rem)$ to be about three minutes, to accommodate the time needed to completely remove the tamper seal. (We want to err on the side of longer time windows if possible, as we can capture attackers who are quicker than expected, but not those who are slower (see Section 6.7).
- We set $w(rem, rep)$ to be around 4-6 hours, as attaching and wiring the extra device to the meter may take some time.

The benign case E_b would feature the same time windows, but with one extra addition:

- The time window between $w(rep, s)$ is defined by the schedule of the incident ticket, since the service must take place within the time set by the schedule. (This may cause issues if a legitimate service call takes longer than expected—for example, the technician runs into unexpected delays while either traveling to or fixing the device—but the schedule can be adjusted in these cases.

Every indicator is binary in this instance, so thresholds can be set to 1 for all of them. Our responses for the events can be set as follows:

Malicious Addition: Here, we want to disable maintenance mode on the device, and perhaps schedule a technician to inspect the device in person.

Scheduled Maintenance: This event is not considered suspicious, so we just log that the service was completed.

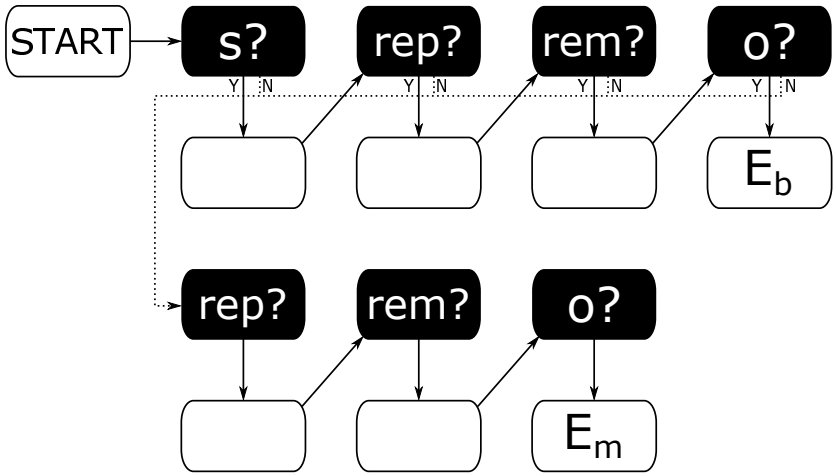


Figure 8.3: A diagram of the factor graph sequence used to represent the Maintenance Attack, as well as its benign counterpart.

Given the indicators, time windows, and responses, TEDDI generates a pair of factor graph sequences (Figure 8.3) to cover both E_m and E_b.

TEDDI then constructs our logical predicates as follows:

- The benign maintenance event E_b is defined by the sequence o, rem, rep, s,[3] where o represents the opening of the cabinet, rem represents the removal of the tamper seal, rep represents the replacement of this seal, and s represents the case where device service was scheduled for the current time. Likewise, the malicious USB event E_m is defined by the sequence o, rem, rep.

- If there exists time periods $S' \in S$, $Rep' \in Rep$, $Rem' \in Rem$, and $O' \in O$ such that they occur in the proper order and within the proper time windows, then we say that the event E_b is occurring. If E_b is not occurring, however, and there exists time periods $Rep' \in Rep$, $Rem' \in Rem$, and $O' \in O$ such that they occur in the proper order and within the proper time windows, then we declare that the event E_m is occurring.

  From here, TEDDI generates the necessary logical predicates:

[3] We use commas here to clearly show the indicators involved.

$$
\begin{aligned}
&\exists\, S' \in S \wedge \\
&\exists\, Rep' \in Rep \wedge (s(Rep') \leq s(S') \leq (e(Rep') + w(rep, s))) \wedge \\
&\exists\, Rem' \in Rem \wedge (s(Rem') \leq s(Rep') \leq (e(Rem') + w(rem, rep))) \wedge \\
&\exists\, O' \in O \wedge (s(O') \leq s(Rem') \leq (e(O') + w(o, rem))) \;\Rightarrow E_b
\end{aligned}
\tag{8.3}
$$

$$
\begin{aligned}
&\exists\, Rep' \in Rep \wedge \\
&\exists\, Rem' \in Rem \wedge (s(Rem') \leq s(Rep') \leq (e(rem') + w(Rem, rep))) \wedge \\
&\exists\, O' \in O \wedge (s(O') \leq s(Rem') \leq (e(O') + w(o, rem))) \;\Rightarrow E_m
\end{aligned}
\tag{8.4}
$$

Finally, TEDDI takes the above sequences and turns them into code for its TIPs and TDPs.

8.5 Scenario 4: Malicious USB Attack

We previously described this attack in Chapter 7, so we only summarize and formalize the attack here:

Indicator Sequence: Both an attacker and a legitimate technician start by *opening* (o) the cabinet door, letting *light* (l) reach the device, *removing* (r) a USB plug, and plugging in a *USB device* (u). The key difference again lies with the explicit authorization of the utility, and we can therefore differentiate the two scenarios with an external indicator saying whether or not the update is *scheduled* (s). Due to the similarities between the sequences, we want to rank the benign event higher than the malicious one, as ranking the malicious event first means we would declare the event malicious before getting a chance to check s.

While some of the early indicators may seem redundant, they help to identify exactly how the event took place (allowing us to tailor our response towards a specific action) and guard against false positives. If the sequence consisted of just u and s, for example, a grid defender would not only be left in the dark as to how the attacker accessed the device, but if the USB device started reporting bad data, the grid defender would have no way to distinguish that from an active attack. The early indicators allow defenders to tell when things don't look quite right (such as when a USB device is detected despite the device's cabinet having never being opened), and let them know exactly how an attacker got to the device and allowing them to set up their response sequence accordingly.

The monitor thresholds are simple to set in this scenario, as they are all binary (Is the USB port uncovered? Is this device scheduled to be updated?) aside from the light sensor, which can be measured to ensure a sudden increase in ambient light will trigger the indicator.

Time Windows: As far as the timing windows are concerned, we assume that a legitimate technician is probably under some amount of time pressure, and won't want to spending more time updating the device than is necessary, so we can set that sequence's time window to 3-4 minutes at absolute most. For the attacker, we assume that they will want to mimic the appearance of a legitimate user as closely as possible, and thus we can use the window settings from the benign case for the malicious case as well.

Responses: For the Malicious USB Attack, we want to log the attempted attack, block and disable the USB port that was used, and monitor the traffic from the device to make sure it does not do anything suspicious. In the benign case, however, we just want to log that the service was performed, and tell our external database to remove it from the schedule.

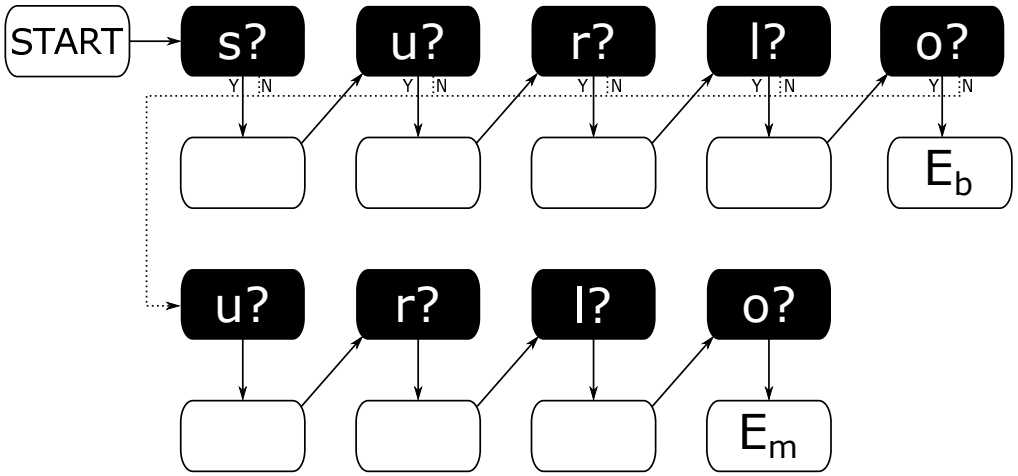


Figure 8.4: A diagram of the factor graph sequence used to represent the Malicious USB Attack and its benign twin.

Graph Sequence: Figure 8.4 shows our factor graph sequences for our malicious USB event E_m and the benign USB update E_b.

Logical Sequence: The benign USB event E_b is defined by the indicator sequence $olrus$, while the malicious USB event E_m is defined by the indicator sequence $olru$. Much like that Maintenance Mode Attack, if there exists time periods $S' \in S$, $U' \in U$, $R' \in R$, $L' \in L$, and $O' \in O$ such that they occur in the proper order and within the proper time windows, then we say that the event E_b is occurring. Otherwise, if there exists time periods $U' \in U$, $R' \in R$, $L' \in L$, and $O' \in O$ such that they occur in the proper order and within the proper time windows, then we declare that the event E_m is occurring.

TEDDI arranges our predicates for E_b and E_m like so:

$$
\begin{aligned}
&\exists\, S' \in S \wedge \\
&\exists\, U' \in U \wedge (s(U') \leq s(S') \leq (e(U') + w(u,s))) \wedge \\
&\exists\, R' \in R \wedge (s(R') \leq s(U') \leq (e(R') + w(r,u))) \wedge \\
&\exists\, L' \in L \wedge (s(L') \leq s(R') \leq (e(L') + w(l,r))) \wedge \\
&\exists\, O' \in O \wedge (s(O') \leq s(L') \leq (e(O') + w(o,l))) \;\Rightarrow E_b
\end{aligned}
\tag{8.5}
$$

$$
\begin{aligned}
&\exists\, U' \in U \wedge \\
&\exists\, R' \in R \wedge (s(R') \leq s(U') \leq (e(R') + w(r,u))) \wedge \\
&\exists\, L' \in L \wedge (s(L') \leq s(R') \leq (e(L') + w(l,r))) \wedge \\
&\exists\, O' \in O \wedge (s(O') \leq s(L') \leq (e(O') + w(o,l))) \;\Rightarrow E_m
\end{aligned}
\tag{8.6}
$$

From here, the predicates are encoded in our TIPs and TDPs.

8.6 Summary: Scenarios 3-4

Our breakdown of the prior two scenarios addresses the following points of the grid defender's dilemma:

Current protection systems lack the power and/or context to differentiate between important tamper events. TEDDI overcomes this hurdle because its distributed setup improves our information-gathering capabilities over a single-device tamper system, and allows us to capture the presence of distributed and context-dependent events.

In these scenarios, for example, the two events are very similar, and only differentiated by an external event. Most tamper protections systems do not have the capability to gather the necessary contextual information to differentiate these two events, and thus must choose arbitrarily how to treat these events when they occur. TEDDI, on

the other hand, is able to gather the necessary context (in these cases, querying an incident database to learn about scheduled service), allowing it to make a proper decision for the situation.

Current protection systems treat any sort of tampering as malicious. Other protection systems are unable to tailor their response based on whether an event is benign or malicious, and would have to treat both the same way. TEDDI, however, has the power to enact the proper response to either case (here, we can choose from information alerts or more active responses like network filtering), and the defender does not have to risk ignoring an attacker or unnecessarily reducing grid availability.

Current protection systems have either no response or a single response. Either a system can only detect when an event is happening (and thus gives an attacker a window to act before a grid defender can respond), or they have a single response to handle any sort of tampering (meaning the defenders risks overreacting to a benign event). TEDDI, as discussed in the previous point, gives a grid defender the flexibility to craft enough responses to handle events in whatever manner is needed.

8.7 Scenario 5: Taum Sauk Dam Overflow

For our fifth case study, we examine the structural failure at the Taum Sauk hydroelectric facility. While TEDDI may not be able to stop this incident using the facility's 2006 monitoring setup,[4] the breakdown of the failure suggests that we can cover the accident using a three-step indicator sequence that includes a regional indicator as its closing step. We start by making a slight change to the facility's pre-failure transducer setup, where the three transducers are enclosed inside a single protective pipe [62]. Instead, we *distribute* transducers around the reservoir to gain a wider view of what the water level may be. We

[4] Admittedly, we have the benefit of hindsight when approaching this event. Had the event not occurred in the first place, we may not have had such a clear idea of how to sense it.

then construct our event sequence as follows:

- Our monitors include the existing transducers (now in new locations) and a sensor placed on the cables holding the transducer pipes into place (to tell us whether the pipe is secured or free). Each transducer setup is assigned a TIP.
- Our local indicators include "Cable Loose" and "Normal Water Level." We also have a regional indicator "High Water Level," for detecting if the majority of sensors see the water level as too high.
- Our exact sequence for our event E_w is as follows: "Cable Loose" (c), then "Normal Water Level" (n), and then our regional "High Water Level" (h). This let us capture the event where the transducer pipe has come loose, and a transducer has moved enough to give us falsely-normal readings when the water level is actually dangerously high.[5]

The thresholds for this scenario are easier to set than in the previous case studies: Whether a cable has come loose is a binary event, and the proper reservoir water level has already been determined. (These levels were set just short of 1600 feet when the disaster occurred, but the resulting study found that these levels were too high to allow for potential mistakes [62], so they have probably been lowered since then.) The response to this event is similarly straightforward: Stop pumping water to the upper reservoir immediately, and alert facility staff that one of the sensors requires maintenance.

For our time windows, we start by setting our window between the loose cable and the local water level reading to be very long (on the order of months, if not years), as it may take a long time for the transducers to rise to an unsafe level one it breaks loose. On the other hand, the window between the normal local and abnormal regional readings will be very short (a few seconds), as the local level will not stop reporting erroneous water levels

[5]Note that this is not actually a case of monitor failure; rather, the monitor's readings are wrong because of incorrect assumptions—i.e., the sensor has been moved without its knowledge.

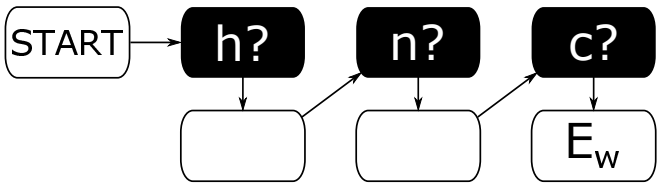


Figure 8.5: A diagram of the factor graph sequence used to represent the Taum Sauk Dam Overflow.

once it starts (and thus the normal local and abnormal regional readings will be present simultaneously).

Our factor graph sequence for this event is shown in Figure 8.5.

Our logical equation breaks down like this: The event E_w is defined by the indicator sequence cnh, and state that E_w occurs when there exists time periods $H' \in H$, $N' \in N$, and $C' \in C$ such that they occur in the proper order and within the proper time windows.

$$\begin{aligned} &\exists\, H' \in H \wedge \\ &\exists\, N' \in N \wedge (s(N') \leq s(H') \leq (e(N') + w(n,h))) \wedge \\ &\exists\, C' \in C \wedge (s(C') \leq s(N') \leq (e(C') + w(c,n))) \;\Rightarrow E_w \end{aligned} \tag{8.7}$$

This scenario highlights how TEDDI handles the following points of the grid defender's dilemma:

Current protection systems lack the power and/or context to differentiate between important tamper events. Once again, we see that a regional event, which a number of tamper protection systems do not have the ability to detect, plays an important role in identifying an event. TEDDI's distributed setup gives it the upper hand over prior work by allowing the user to collect the contextual information they need.

Current protection systems treat any sort of tampering as malicious. The overfilling of the Taum Sauk upper reservoir was a mechanical failure, not an attack, and responding to it as if an active adversary were in the system (for example, isolating

the facility from the grid) could reduce the availability of this system longer than needed. TEDDI, on the other hand, allows the grid defender to define and handle non-malicious events (unlike many other protection systems), and the problem can be pinpointed and fixed with minimal downtime.

8.8 Other Tamper Scenarios

For the sake of completeness, we offer a brief summary of how TEDDI handles the remaining scenarios from Chapter 3, including the indicators, graph sequences, and logical equations that are involved.

8.8.1 Simple User Data Heist

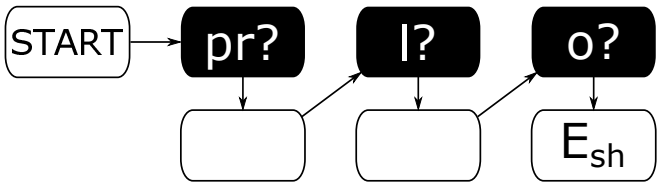
Indicator Sequence: An attacker must first *open* (o) the case of the device, let *light* (l) into the device as they locate the memory chip with the data they want, and then place *probes* (pr) onto the chip to collect the data when it appears.

Monitors: We will need a cover switch or seal to detect when the device is opened, a photosensor to catch the light that reaches the device's circuitry, and a probe sensor, such as a piezo-electric sheet [151], to alert when a probe has been placed on the device.

Time Windows: The windows for this sequence will be relatively short: We expect the light to stream in soon after the case is cracked open, and while the probes make take some time to place, the attacker will be motivated to finish the attack quickly, as they do not need to be present as the probes wait for the desired information. Therefore, we choose to set the open-light window at sixty seconds, and the light-probe window at ten minutes.

Responses: For our response, we choose to log the event, flag the device for maintenance, and hold off on having the device report its data back to the control center (or perhaps only report a subset of this data until the device is confirmed to be safe).

Graph Sequence: The graph sequence for our event E_{sh} is as follows:



Logical Sequence:

$$\begin{aligned} &\exists\, Pr' \in Pr \wedge \\ &\exists\, L' \in L \wedge (s(L') \leq s(Pr') \leq (e(L') + w(l, pr))) \wedge \\ &\exists\, O' \in O \wedge (s(O') \leq s(L') \leq (e(O') + w(o, l))) \;\Rightarrow E_{sh} \end{aligned} \tag{8.8}$$

8.8.2 Complex User Data Heist

Indicator Sequence: An attacker must first apply *X-rays* (x) to imprint the desired data, *open* (o) the device to access the memory chip, let *light* (l) into the device as they locate the appropriate chip, and then *remove* (r) the chip to harvest the data.

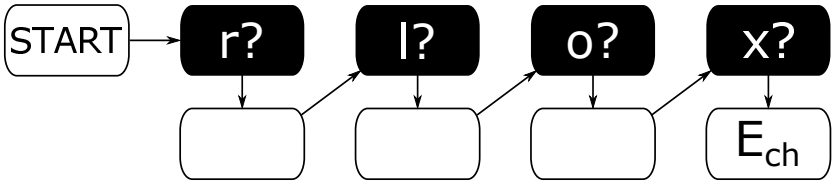
Monitors: Here, we need a radiation sensor to detect when an attacker bombards the device with X-rays, a cover seal and photosensor to know when the device itself has been breached, and finally a seal on the memory chip itself that alerts our system when it is breached or removed.

Time Windows: Imprinting data via X-rays may take some time, so we set the time window between the radiation ending and the device being opened to be six hours. Light will stream in quickly after the device is opened, so this time window is set to sixty seconds. Finally, accessing the memory chip may prove to be a difficult task

(depending on the strength of the seal used to secure it), so we set the window at four hours to avoid missing a slower attacker.

Responses: This is a highly invasive attack, and it warrants a severe response. Because the sensitive data is imprinted into the chip by the attack, our best course of action is to physically destroy the chip containing the memory. (Weingart suggests using thermite to incinerate the chip [151], but grid defenders would have to ensure that the reaction is contained enough to not cause collateral damage.)

Graph Sequence: The graph sequence for our event E_{ch} is as follows:



Logical Sequence:

$$\begin{aligned}
&\exists\, R' \in R \wedge \\
&\exists\, L' \in L \wedge (s(L') \leq s(R') \leq (e(L') + w(l,r))) \wedge \\
&\exists\, O' \in O \wedge (s(O') \leq s(L') \leq (e(O') + w(o,l))) \wedge \\
&\exists\, X' \in X \wedge (s(X') \leq s(O') \leq (e(X') + w(x,o))) \;\Rightarrow E_{ch}
\end{aligned} \tag{8.9}$$

8.8.3 Pin-In-The-Meter Attack

The Pin-In-The-Meter attack [132] can be modeled as follows:

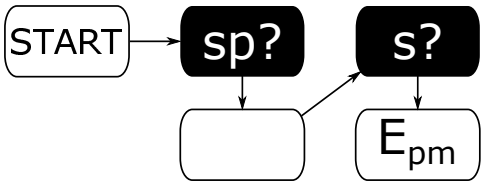
Indicator Sequence: To execute this attack, an attacker must first drill through the edge device's exterior (causing the box to *shake* (s)), and then inserting a pin to cause the wheel inside the meter to stop *spinning* (sp).

Monitors: We can use an accelerometer to detect when the box is being drilled, and a motion sensor to determine if the meter wheel is spinning.

Time Windows: We assume that the attacker will be motivated to stop his or her meter as quickly as possible, as this attack is financially-motivated and the longer the meter is disabled, the more money the attacker saves. We therefore set the time window between the shaking and the meter wheel stopping at two minutes.

Responses: At the very least, we want to note this event and send out a technician to investigate it. Cutting off network traffic (and potentially electric service) to the device is also a possibility, as the availability impact would be limited to the customers using that meter.

Graph Sequence: The sequence for our event E_{pm} would appear like this:



Logical Sequence:

$$\begin{aligned} &\exists\, Sp' \in Sp \wedge \\ &\exists\, S' \in S \wedge (s(S') \leq s(Sp') \leq (e(S') + w(s, sp))) \;\Rightarrow E_{pm} \end{aligned} \tag{8.10}$$

8.8.4 Return-To-Debug Attack

Indicator Sequence: Here, the attacker must *open* (o) the device, allowing *light* (l) to reach the inner circuitry, and then *reconnect* (rec) the debug pins on the circuit board via the ion beam.

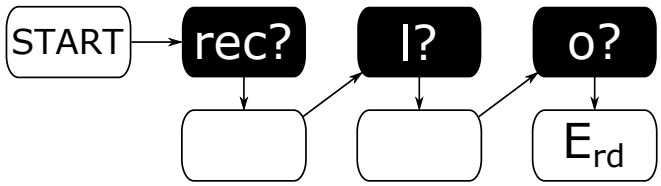
Monitors: A cover switch/seal and photosensor are required, as well as a monitor on the debug pins to detect if and they are activated.

Time Windows: The open-light window will be small (about sixty seconds), while the light-debug window is set to thirty minutes to give the attacker time to properly re-

connect the pins.

Responses: While an attacker may just be stealing data and not changing the behavior of the device, we still want to verify the device's behavior until a technician can inspect it. Therefore, we choose to monitor the device's traffic, and potentially revoke certificates that let the device talk to higher-value targets on the network.

Graph Sequence: We define the graph sequence for our event E_{rd} as follows:



Logical Sequence:

$$\begin{aligned} &\exists\, Rec' \in Rec \wedge \\ &\exists\, L' \in L \wedge (s(L') \leq s(Rec') \leq (e(L') + w(l, rec))) \wedge \\ &\exists\, O' \in O \wedge (s(O') \leq s(L') \leq (e(O') + w(o, l))) \;\Rightarrow E_{rd} \end{aligned} \tag{8.11}$$

8.8.5 The Sensor Subversion Scenario

Indicator Sequence: Here, the attacker *shakes* (s) the box with its drilling, avoids the cover switch, lets *light* (l) into the box, and *disconnects* (d) the edge device.

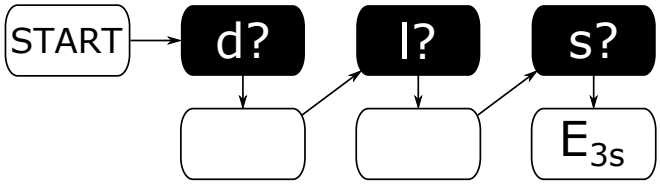
Monitors: For this sequence, we need an accelerometer, photosensor, and network cable monitor to detect this sequence. (We assume we have a cover switch as well, if for no other reason than for the attacker to target it.)

Time Windows: The shake-light time window will be much longer than in the Schweitzer Scenario (i.e. 3-4 hours, if not longer), as the attacker will have to apply the glue to the cover switch by aiming through a small hole, and then wait for the glue to set and disable the switch. Once the attacker opens the box, however, they will move

quickly to plug in their own device, and thus the light-disconnect window will be short (perhaps five minutes at most).

Responses: Much like in the Schweitzer Scenario, the traffic coming from the edge device's access point will be either filtered or dropped, and the certificate used by the edge device for authentication may also be revoked.

Graph Sequence: We define the graph sequence for our event E_{3s} as follows:



$$\begin{aligned} &\exists\, D' \in D \wedge \\ &\exists\, L' \in L \wedge (s(L') \leq s(D') \leq (e(L') + w(l,d))) \wedge \\ &\exists\, S' \in S \wedge (s(S') \leq s(L') \leq (e(S') + w(s,l))) \;\Rightarrow E_{3s} \end{aligned} \tag{8.12}$$

8.8.6 Earthquake

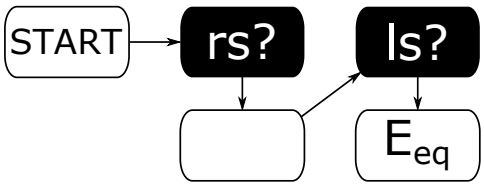
Indicator Sequence: Here, we look for *local shaking* (ls) at the edge device level, and then check to see if *regional shaking* (rs) is present (i.e., many devices are shaking). (For this event, we define "shaking" as any acceleration values that rate a four (IV) or above on the Mercalli scale [137].)

Monitors: We only need an accelerometer here, as both the local and regional indicators in our sequence are based on its readings.

Time Windows: The shaking at all of the devices should occur at roughly the same time, so the time window between LS and RS will be very small (five to ten seconds).

Responses: For this event, we want to alert the utility to its presence, and perhaps initiate a disaster response protocol to prepare for potential service losses.

Graph Sequence: We construct our graph sequence for the event E_{eq} as follows:



Logical Sequence:

$$\begin{aligned} &\exists\, RS' \in RS \wedge \\ &\exists\, LS' \in LS \wedge (s(LS') \leq s(RS) \leq (e(LS') + w(ls, rs))) \;\Rightarrow E_{eq} \end{aligned} \tag{8.13}$$

8.9 Overall Summary

The above scenarios demonstrate how TEDDI (nearly) solves the grid defender's dilemma:

- TEDDI can gather the necessary context information needed to differentiate between important tamper events.
- TEDDI does not force us to treat any tampering as malicious.
- TEDDI provides a flexible response strategy to allow operators to deal with events with exactly the right amount of severity.
- TEDDI can intervene earlier in the kill chain [54] of an attack than many other protection systems.
- TEDDI reduces the amount of manual configuration needed to define a protection system for any arbitrary SCADA system.

However, there is one final point in the dilemma that we need to consider: Does TEDDI adhere to the strict performance constraints of the power grid? We address this question, as well as take a closer look at the amount of effort required to configure and use TEDDI, in the next chapter.

Chapter 9

Evaluation

In this chapter, we evaluate TEDDI's performance in three areas: speed, accuracy, and configuration resource requirements. In each case, we find that TEDDI has equal or better performance than existing tamper solutions, all while adhering to the strict constraints of the power grid.

9.1 A Word on System Comparison

In looking through the prior work on this subject, a concerning trend emerges: The speed and performance evaluations of these systems are not always rigorous, and are sometimes non-existent (and in the case of usability analyses, *always* non-existent). Table 9.1 offers a sampling of prior evaluations.

While we cannot definitively say why evaluations in this area are so spare, we can offer some theories:

- Some of these systems are proposals (for example, EBTA [146] and CAPMS [138]), and do not have a proof-of-concept implementation to evaluate.
- In our discussions with industry representatives, we found that they were loathe to

Table 9.1: A sampling of some of the evaluation results from prior protection solutions.

System	Accuracy	Performance
RRE [161]	None	Calculates optimal response actions for 900-node tree in under 45 secs; generates model for 330K nodes in under 24 ms
EBTA [146]	None	None
PAC [141]	Detected 1 of 1 attacks	None
SCADA-Hawk [123]	Detected 1 of 1 attacks	None
Amilyzer [14]	Detected 4 of 4 attacks	No hard numbers, but scaled to 32K nodes in real meter deployment
PQS [105]	Detected 1 of 1 attacks	When facing a DOS, takes 30 secs to switch to 'Questionable,' and 19 secs to then switch to 'Compromised'
SCPSE [162]	Detected 3 of 3 attacks	Took about .08 secs per state to locate bad data; upper bound of about 11.7 secs for finding bad data in each state; generates an attack graph template for 3500 nodes in roughly 1200 ms
Edwards [39]	89.75% accuracy against three intruder platforms	None
Kenaza et al. [64]	Roughly 76% accuracy against six attacks after 16 test iterations	None
Boggs et al. [17]	0.03% FP rate; true positive rate unknown (no ground truth)	Average of 4,579.85 minutes between similar alerts from separate sites
Neves and Araujo [85]	No attacks considered	Cost classified as free/cheap
Collberg et al. [29]	Blocked 8 of 8 attacks	Code breakdown adds 5-20% overhead; latency cost dominated by compilation ($\approx$ 1 sec per function)
ADSDB [8]	Defeated 4 of 5 attacks	Only production costs considered
GCK [43]	Defeated 3 of 3 attacks	Overhead ranged from 90–969 μs
TLP [131]	Resisted 4 of 4 attacks	Added 1 sec to bzip runtime in slowest configuration
IBM 4758 [120]	Defeated 7 of 7 attacks	Immediate secret zeroization
Dragone [37]	Detected 1 of 1 attacks	None
Megalingam et al. [78]	Detected 1 of 1 attacks	Tamper detection speed not tested
Desai [36]	Defeated 2 of 2 attacks	Speed not evaluated, but hardware overhead $\leq$ 10% for each technique
Patagonix [72]	Defeated 9 of 9 rootkits	Averages $\leq$ 160 μs to identify code; imposed up to 30% overhead on apps
SRID [145]	Detected 23 of 24 attacks and 49 of 56 attack origins	None
Autoscopy	Detected 15 of 15 rootkits [100]	Imposed less than 5% overhead [99]
Roosta et al. [106]	Defeats 6 of 6 attacks	None
Düssel et al. [38]	Detected 8 of 8 attacks; $\leq$ .1 avg FP rate on 13 others	Throughput up to 429.1 Mbps
CAPMS [138]	None	None
BRIDS [81]	26 attacks considered across 3 devices; at least 92% detection accuracy per device	None

discuss specific examples of attacks on their systems, and consequently real examples of attacks are very hard to come by. This may explain the limited number of attacks used for evaluation in prior work.

- Performance evaluations are often considered secondary to accuracy results, since a fast system that does a poor job of detecting attacks is of no use to anyone. This attitude, however, is counterproductive in a power grid setting, where performance is just as important (and perhaps more important) than accuracy.

Overall, prior work in this space does not provide a great basis to use for comparison with TEDDI. Therefore, our evaluations will instead be geared towards answering the following questions:

1. **Can we achieve a correct event decision rate of 90% or better?** We derive this rate from the accuracy rate of BRIDS [81] and the false-positive rate of Düssel et al. [38].

2. **Can we process our factor graphs and come up with an event decision in under 400 μs?** This number comes from IEEE Standard 1646-2004 [56], where several types of substation communication have an upper bound of 8 ms on delivery time, and from Autoscopy Jr. [99], where we state that we want to incur less than 5% overhead (400 μs is 5% of 8 ms). (Unlike our indicator windows, factor graph processing time may get in the way of an edge device's primary function, which is why we want to keep this time as short as possible.)

In addition, we wish to examine the amount of effort required to configure our system, an attribute that is not considered by any of the prior work in this space. To accomplish this task, we examine a sample case study to see how easy it is to translate a problem within the narrative into a workable protection system using TEDDI, as opposed to using other similar protection solutions. More specifically, we attempt to answer the question **"Given**

a small set of events, can a grid defender create a TEDDI system that detects these events with less time/resources than if we used another system?"

With our three questions in hand, we now evaluate TEDDI to see how well we can answer them.

9.2 Detection Accuracy

Because of our distributed setup, we need to evaluate TEDDI's accuracy at both the TIP and the TDP levels.

9.2.1 TIP Event Detection

We set up our TIP testing as follows:

- We configured a single TIP to read from ten different monitors, with each monitor containing either a zero or one. All of the monitors were simulated using text files, mimicking Linux's habit of representing external devices as files in the `/dev` directory.
- The TIP was given a ten-function-node[1] factor graph, where each node in the graph represented a unique indicator. In alignment with the monitors, each indicator had a threshold of one, and were said to be present if the corresponding monitor contained that value.
- The graph itself contained four events: Three local events defined by three-indicator sequences that could be definitively identified by the TIP, and a fourth event defined by a single regional indicator. The time windows for the local sequences ranged from

[1]While factor graphs have both function and variable nodes, recall that variable nodes are mostly placeholders in our system, and thus do not incur any computational cost.

seven to sixty seconds. The regional event was the lowest-priority event of the four, and while it was primarily targeted at our TDP testing, it was still important to know if the TIP would recognize the possibility of a larger event and properly defer to the TDP.

- The TIP test consisted of one hundred rounds, each of which lasted ten seconds. In each round, for each monitor, we randomly select a value (either 0 or 1) corresponding to whether or not the given indicator will be present or absent in this round. Our data generation model can be considered a zero-order approximation, in which both 0 or 1 are equally likely to be selected [94]. We choose this model rather than a larger-order approximation because we do not have any ground truth data on how often these indicators appear in real environments, and thus cannot make any assumptions about how likely an indicator is to appear.

  We obtain our random data by drawing a byte from `/dev/urandom`, dividing by two, and placing the remainder into the proper monitor file, and we repeat this process for all of the system's monitors. These files were then read at three-second intervals by the TIP, meaning that the files could be read three or four times in a single round. This meant that the TIP's event decisions could change even when the monitor data did not, as indicators that are not currently present may eventually fall out of the prescribed windows.

  For example, consider the indicator sequence AB, where the allowed time window between A and B is 5 seconds. If a round change causes A to flip from present to absent and B to flip from absent to present, the first two the TIP reads the data (assuming the TIP had been reading the data prior to the change, and reads the data immediately after it changes), the event will be considered present because A was last seen 3 seconds ago. The next time the data is read, however, the event will be considered absent because A's last sighting would now be six seconds ago.)

- The data from each round and decision were manually verified after the tests to determine what the correct event decision should be, and see how well the TIP performed.

Over the course of the one hundred rounds, the TIP made a total of 332 event decisions. Of these, 329 of the decisions matched our manual analysis, giving us an accuracy rate (99.1%) that far exceeds our 90% goal. Our incorrect decisions were concentrated in a single round, meaning that we achieved 100% accuracy in ninety-nine of our one hundred rounds.

While these numbers are more than satisfactory, they beg the question: What happened in that one round (Round 71, to be specific) that threw off our decision engine? It turns out that this round exposes a limitation in our TIP prototype, one whose solution could create a significant performance issue.

The Curious Case of Round 71

First, for the sake of clarity, we label our events and sequences as follows:

- Our highest-priority event is labeled *Event 1*, and is defined by the indicator sequence ABC.
- Our second-highest priority event, *Event 2*, and is defined by the indicator sequence DEF.
- Our third-highest priority event, *Event 3*, and is defined by the indicator sequence GHI.
- Finally, our regional event *Event R* has the the lowest priority, and is defined by the regional indicator J. If none of the other events are present, the TIP must therefore send an alert to the TDP and let it decide if R is present.

At the start of Round 71, the TIP looked to see if the final indicator of any of these sequences were currently present. Indicators C and I were absent for this round, which left events 2 and R as the only possible decisions.

Indicator F was present in Round 71, but D and E were not, leading TEDDI to consult its history counters. The maximum-allowed time window for F and E was twenty-five seconds, and the history counters showed that two prior instances of E fell within this window: one at Round 70 (3-12 seconds before 71) and one at Round 68 (24 seconds).

The error arose in TEDDI's selection of the instance of E to use for event detection. Our prototype was constructed to select the *most recent* instance of an indicator when looking for event sequences, which meant that TEDDI selected the E from Round 70 to continue its search from. However, the most recent occurrence of D happened in Round 67, and the maximum time window allowed between D and E was only ten seconds. This led TEDDI to conclude that Event 2 was not present, and it asked its TDP to look for Event R. However, had TEDDI continued searching using the E instance from Round 68, it would have found that D fell within the proper time window, and made the correct decision that Event 2 was present.

The simple solution to this problem is to have TEDDI look for events using *every* viable indicator instance, rather than picking just one. However, this sort of setup leads to a potentially exponential increase in the number of indicator instances to check: After checking the last indicator in a sequence, our last-five history counter setup means that there could be five instances to check for the next indicator, which could mean checking twenty-five instances of the indicator beyond that, one hundred twenty-five beyond that, and so on. While both our sequence chains and time windows were relatively short for our accuracy tests, having longer sequences and larger time windows increase the chances of seeing the full exponential case.

Given the potential for such a solution to lead to performance issues in an industry that is particularly sensitive to them, we decided that are most-recent-viable-indicator setup was preferable to checking every viable indicator. However, the latter option could be made more palatable by reducing the number of history counters we maintain (for example, cutting down from five to two or three). Taking another approach with our algorithm may also help mitigate this problem (for example, treating our graph as a state machine and moving through it in real-time as indicators appear and disappear), but they would require substantial changes to our data fusion structure.

The Danger of Too Few Time Periods

To test our theory about longer time window increasing our chances of seeing failures like Round 71 above, we re-ran our accuracy tests with some small changes:

- The indicators and overall structure of our factor graph were not changed, but we lengthened the time windows between indicators. For this test, they ranged from ten to one hundred eighty seconds.
- We altered our round length to match our TDPs tests (see Section 9.2.2). Each round now lasts three seconds, giving the TIP an entirely new dataset every time it checks its monitors. We also increased the number of rounds from 100 to 200 to see how our TIP held up under long periods of rapidly-changing data.

Our TIP proved to be remarkably resilient to our changes: Despite the higher number of rounds, longer time windows, and increased data variability, the TIP made the correct event decision in 199 of 200 rounds (99.5%). Interestingly enough, our one incorrect decision (once again involving Event 2 and its DEF event sequence) came about for an unexpected reason:

- At the time of our incorrect decision in Round 62, our F indicator had been present since Round 56. The first available time period for E that met our sequence criteria (i.e., it started before Round 56, but within the sixty-second window allowed between E and F) was also fairly long, stretching from Rounds 46 to 55.

- Indicator D had been present in Round 43, which fell well within the available time window (in this case, 30 seconds before Round 46). However, in between Rounds 44 and 62, D had appeared and disappeared five separate times, meaning that D's presence in Round 43 was no longer stored in our history counters by the time we reached Round 62.

- Because of our history counter limitation, TEDDI missed the presence of Event 2, and instead reported the presence of the lower-priority Event 3. If TEDDI had kept six time periods in its history counters instead of five, the system would have been able to capture Event 2's presence.

This result suggests that we should *increase* the number of time periods we collect, contradicting the conclusions we drew in our first test! Ultimately, the optimal number of time periods required for event detection will depend on the environment of the edge device (for example, do we expect indicators to appear and disappear rapidly near an edge device?), which suggests that the number of time periods should be a configurable parameter in future iterations of our prototype.

Overall, however, the TIP make 528 correct event decisions in 532 opportunities, giving us a final accuracy rate of 99.2%, far above the 90% threshold we sought to beat.

9.2.2 TDP Regional State Calculation

Since we had already evaluated factor graph accuracy in our TIP tests, we decided to focus on the regional state calculations of the TDP, as this feature is the primary difference be-

tween our full and limited graphs. We used the same monitors, indicators, and factor graph as with the TIP tests, so the TDP's decision on whether or not Event R was present was completely dependent on its ability to accurately calculate the regional state across all of its TIPs.

For this test, we connected four TIPs (via wired network connections) to a single TDP. This time, the test rounds were only three seconds long, potentially making the data different every time the TIPs read from the monitors. Only the local indicator tied to our regional indicator J received random data. Every other indicator was absent, ensuring that the TIPs would keep going back to the TDP for assistance.

We ran our TDP test for fifty rounds, during which the TDP made 200 decisions on the regional state of the TDPs. Afterwards, the results were again manually verified to validate the TDP's decisions. We found that the TDP correctly calculated the regional state, and therefore made the correct event decision, in all 200 instances.

An important things to note here is how TIP synchronization (or lack thereof) affects our regional state calculations. Ideally, the TIPs all collect and send their data at roughly the same time, so that when a coordinated regional event occurs (such as an earthquake), all of the data arrives within the 100 millisecond waiting time of the alert, and each TIP sees the proper regional state. Otherwise, if the alerts are received one at a time, a regional event will not be reported until half of the TIPs have sent in their data. (Network latency can also cause problems, as alerts that have to travel over slower or more-congested networks may not arrive within our waiting time. However, such latency would have to be persistent and widespread to affect our calculations.)

9.3 System Performance

Next, we investigate the performance impact of TEDDI at the TIP, TDP, and Generation Tool levels.

Factor Graph Performance

For the factor graph performance tests, we used the following hardware:

TIP: We used the Raspberry Pi 2 Model B [98] for our SCADA edge devices. We based this decision on a conversation with one of our industry contacts, who stated that they use Pis for edge device prototype development.

TDP: We used a a Dell Precision 340 desktop computer [34] equipped with a 2 GHz Pentium 4 processor and 1 GB of RAM, and running version 12.04 of the Ubuntu operating system. While this machine is less powerful than the substation computers available today (for example, the SEL 3355 can have either a dual-core 2.5GHz or quad-core 2.1GHz processor as a base [111]), given the prevalence of weaker legacy devices in the grid, we decided that using an early-2000s-era machine was acceptable.

In the TIP test, we placed a TIP on three different Pis, connected them all to a single TDP (again via wired connections), and had each TIP go through ten rounds of decision making to see how long it took to traverse their factor graphs. For the TDP tests, we just had a single TIP go through 30 rounds of decision making.

We began by using the factor graph from Section 9.2.1 for our performance tests, which meant that the full and limited factor graphs had the same number of nodes (the difference being that the TDP had enough information to make a decision upon reaching the last node, whereas the TIPs did not). From here, we increased both the number of events and

Table 9.2: A table of the factor graph processing times for both TIPs and TDPs.

Graph Size	TIP Performance	TDP Performance
10 Nodes	111.083 μs	53.338 μs
26 Nodes	138.187 μs	70.668 μs
50 Nodes	161.461 μs	94.514 μs
99 Nodes	232.586 μs	143.255 μs

the lengths of the indicator sequences in the graph (while still ensuring the full and limited graphs had the same number of nodes), to observe how well our algorithm scaled. In all cases, the TIP monitors were instrumented such that both the TIPs and TDP would be forced to go through every node in their factor graph to make a decision.

We conducted four separate tests, each with a different size of factor graph (10, 26, 50, and 99 function nodes). We feel that these sizes accurately reflect the graphs for real-world networks, as most sequences appear to be fairly short (in fact, the longest sequence we built in Chapter 8 was five function nodes long), meaning that these graph sizes give grid defenders plenty of room to define 20–30 tamper events that they may be concerned about.

The results for the average factor graph processing time across all of the TIPs can be seen in Figure 9.1 and Table 9.2. The processing time was less for smaller graphs, and appeared to increase linearly as the graph size increased. For ten function nodes, our TIP processed the graph in 111.083 μs, while the TDP processed the graph in 53.338 μs. For 26 nodes, the TIP needed 138.187 μs, while the TDP required 70.668 μs. The 50-node graph took 161.461 μs to walk through on the TIP, and 94.514 μs on the TDP. Lastly, the TIP processed our largest graph (99 nodes) in 232.586 μs, while the TDP needed just 143.255 μs. On the whole, both the TIPs and TDP performed well even when processing the larger graph, and kept processing times comfortably underneath our 400 μs limit.

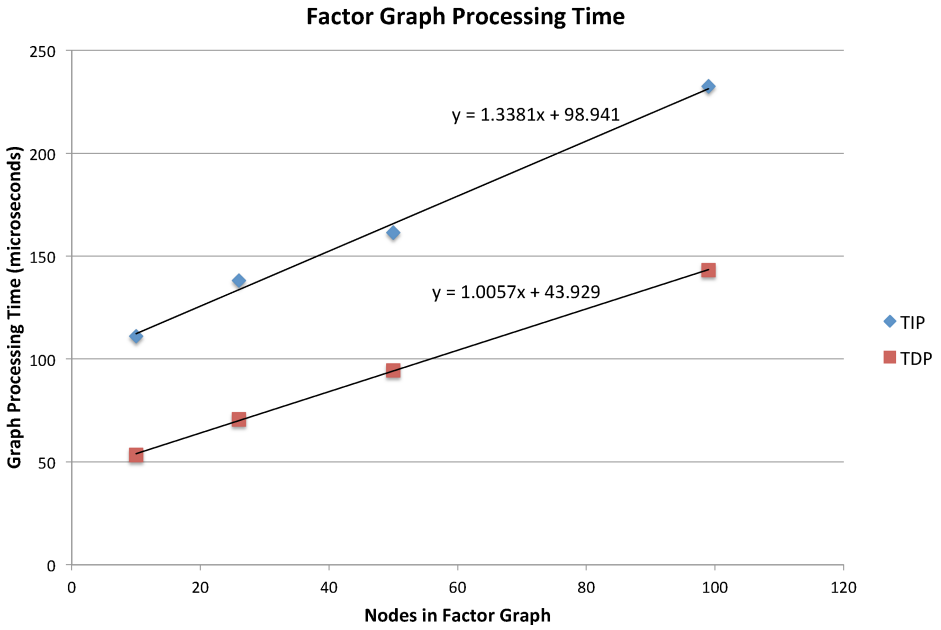


Figure 9.1: Factor graph processing times for both TIPs and TDPs. Note that even in the 99-node graph case, the times fall well within our 400 μs limit.

TDP Placement Performance

While our TDP placement algorithm is run in an offline setting and therefore not subject to the real-time performance demands of the power grid, it is still important that the algorithm do its job within a reasonable amount of time. To verify that this is the case, we tested our algorithm against a number of different network topologies to see how well it performed.

We used three distinct topology types in our test runs:

Linear: The network nodes are arranged more or less in a single line. We chose to test this topology because we received an example of a real substation network that was arranged in this fashion.

Mesh: The network nodes are arranged in a grid pattern, and each node is connected to its immediate neighbors. We took this topology choice from Ward [148], who cited "conversations with industry contacts" [148] for choosing this topology for his simulation.

Star: The network nodes are arranged in several tree-like structures, with a single central node serving as the root for each tree. This topology also came from Ward [148] for the same reasons as the mesh network.

We ran our tests using the Firefox Web Browser on two different machines: The Ubuntu desktop machine used in Section 9.3, and a MacBook Pro running OSX version 10.10 with a 2.4 GHz processor and 4 GB of RAM.[2] Each topology was tested at five different sizes, and the time for each size/type combination was averaged over ten runs. The results of our testing are shown in Figure 9.2 and Table 9.3.

Linear and mesh networks exhibited no difference in their placement times, with the largest of the networks (a 128-node linear network) requiring only .583 seconds to deter-

[2]The browser versions for the original tests were not recorded; however, a re-run of selected topologies using browser version 45.0.1 on both Ubuntu and OSX generated similar times, and we saw no significant difference between times across machines in either the original or subsequent tests.

mine where TDPs should live. Star networks, on the other hand, took a bit more time to process—for example, a 120-node star network needed .749 seconds for TDP placement despite having eight fewer nodes than the linear network mentioned previously. The reason for this difference may lay with the number of TIP nodes in the network, as our star networks tend to have a higher percentage of TIP nodes than the other topologies. (Intuitively this makes sense, given that "leaf" nodes in a star network not only tend to make up the largest group of nodes in the graph, but are also the places we expect edge devices, and therefore TIPs, to live.) TIP nodes require more processing in our TDP placement code (for example, we need to figure out which TDP set to place them in, and then check the entire set to see if any were excluded), and thus having more of them will slow down our algorithm. As evidence, we note that our 84-node star network, which had 75 TIP nodes, and our 128-node linear network, which had 72 TIPs, have very similar processing times (.581 and .583 seconds, respectively).

One important point to note: Our test networks are a bit small compared to real-world SCADA networks, which may explain the linear relationship we found between network size (specifically, the total number of network nodes) and the placement tool's performance (despite our expected quadratic relationship). To test our algorithm on a larger network, we built a model based on the energy management (EM) network here at Dartmouth, which was based on the information gathered from conversations with Dartmouth network operators and features 441 nodes (400 of which are TIP nodes) arranged in a star topology. Despite its size, however, the TDP Placement Tool was still able to calculate the optimal TDP locations for the network in 2.811 seconds, which was only slightly higher than our star-topology trendline predicted (2.752 seconds).

As a final test, we tripled the size of our EM example (1323 nodes) and ran it through our placement tool. This larger network's average TDP placement time (8.790 seconds) is higher than our trendline predicted (8.221 seconds), but is small enough to suggest that a linear model with a steeper slope is still a better fit than a quadratic model. (Indeed, if

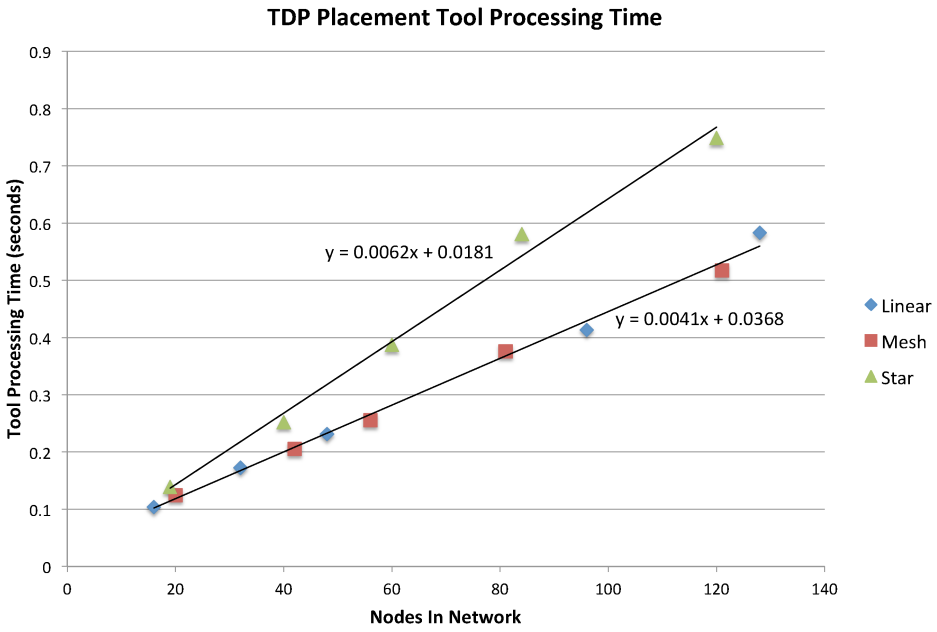


Figure 9.2: TDP placement tool processing times for networks of different shapes and sizes. (Note that our real-life examples are excluded.) As showcased by the trendlines, the star topologies took longer to process than the other networks.

the larger data points are included in our trendline calculations, the star trendline equation shifts slightly to $y = 0.0066x - .0201$.) Further testing may be needed to pinpoint the best growth model, but regardless of whether this trend is a linear or a slow-growing quadratic relationship, the generated times do not suggest that the tool will become noticeably slow for reasonably-sized SCADA networks.

While these numbers fall a bit short of the model generation times mentioned in Table 9.1, they are not onerous, and are perfectly reasonable for a one-time, offline step. Future testing on larger networks, however, may be needed.

Table 9.3: TDP placement processing times, including a large real-life example based on the energy management network at Dartmouth.

Network Type	Network Size (# Nodes)	Average TDP Placement Performance
Linear	16	0.104 s
	32	0.173 s
	48	0.231 s
	96	0.414 s
	128	0.583 s
Mesh	20	0.125 s
	42	0.206 s
	56	0.256 s
	81	0.376 s
	121	0.517 s
Star	19	0.139 s
	40	0.252 s
	60	0.387 s
	84	0.581 s
	120	0.749 s
EM Example (star)	441	2.811 s
	1323	8.790 s

9.4 Usability Analysis

While we would like to provide a measurement of TEDDI's usability using evaluation criteria used by existing methods, tamper and intrusion protection systems often ignore this issue; in fact, ***none*** of the related work in this area considers it in their evaluations. However, some of the prior work in domain-specific languages gives us a blueprint for our own evaluation—namely, they use simple case studies to observe the improvements their languages provide over prior work ([24, 130]).

Along those lines, we now consider a case study describing a tamper incident involving edge devices in critical infrastructure, and see what time and/or data is required to construct a TEDDI system to detect that event, versus building a similar protection system from selected prior work. (While we found that industry members are loathe to discuss specific tamper events that had affected their organizations, this fact does not impact our analysis.)

In addition to TEDDI, we examine the following five systems, which we believe are the works within the SCADA protection space that are most similar to TEDDI:

- the Response and Recovery Engine (RRE) [161],
- the SCADA-Hawk system [123],
- Process Query Systems [105],
- the Probabilistic Alert Correlation system [141], and
- the Security-Oriented Cyber-Physical State Estimation system [162].

For this study, we determine the amount of information required to set up a working protection system against the given event, and see how TEDDI fares in comparison to the field. We begin by defining an example scenario with the following components:

- **Edge Devices:** The operator wants to protect a set of recloser controls mounted within metal cabinets attached to utility poles.
- **Events:** The tamper events we are concerned about are:
    1. The Schweitzer Scenario (Section 8.2),
    2. A Malicious Firmware Attack (Section 8.5),
    3. A Benign Firmware Update (Section 8.5),
    4. An earthquake (Section 8.8), and
    5. A (benign) local shake event (for example, shaking caused by a passing truck).
- **Responses:** We also have the following response possibilities:
    1. Send a response back to the control center alerting them of the event.
    2. Selectively filter network traffic coming from the device.
    3. Passively monitor network traffic coming from the device.
    4. Completely sever the network connection of the edge device.
    5. Revoke the device's credentials to send traffic on the SCADA network.
    6. Delete any secret data (such as cryptographic keys) from the device.
    7. Disable the external USB ports on the device.

    (Note that taking no response is also an option.)

TEDDI: This scenario lines up nicely with the data required by FGDSL. We previously defined the indicators and the sequences for events 1–4 in Section 8, and the shaking events can be quickly defined as just local shaking.

Since we already require an accelerometer and a shaking indicator for the Schweitzer scenario, the local shake event adds nothing extra to either our monitor set or our response set (since we will take no action in response). Thresholds can be set relatively

low or high (or better yet, measured experimentally) to strike the proper balance between false positive and negatives.

Now, we must rank the events to make sure that we always capture the most important ones. We note that similar events need to be ranked such that TEDDI looks for the more specific event first, which means that the benign firmware update should be checked before the malicious one, and the earthquake should be checked before the local shake. With this is mind, we rank the events like so:

1. The Schweitzer Scenario (E_s).
2. The Benign Firmware Update (E_b).
3. The Malicious Firmware Attack (E_m).
4. An Earthquake (E_q).
5. A Local Shake Event (E_l).

Once the events are defined and ranked, TEDDI can turn the data into a suitable factor graph, which is shown in Figure 9.3. (Note that the indicator and event variable are different than in Chapter 8; consult the figure key for more information.)

RRE: Much of the information required by the RRE is the same as TEDDI: indicators, responses, assets to protect, etc [161]. However, there is one major difference: The complete attack response trees (ARTs) used by the RRE have to be built ahead of time (as opposed to TEDDI, which needs the individual sequences but can combine them automatically using its generation tool). For a simple event set, this is not a big problem, but as the events become longer and more complex, a problem emerges:

- The RRE focuses on the overall security properties that the system wants to maintain, and *a separate tree must be maintained for each of these properties.* (The authors suggest using three to cover the classic confidentiality, integrity,

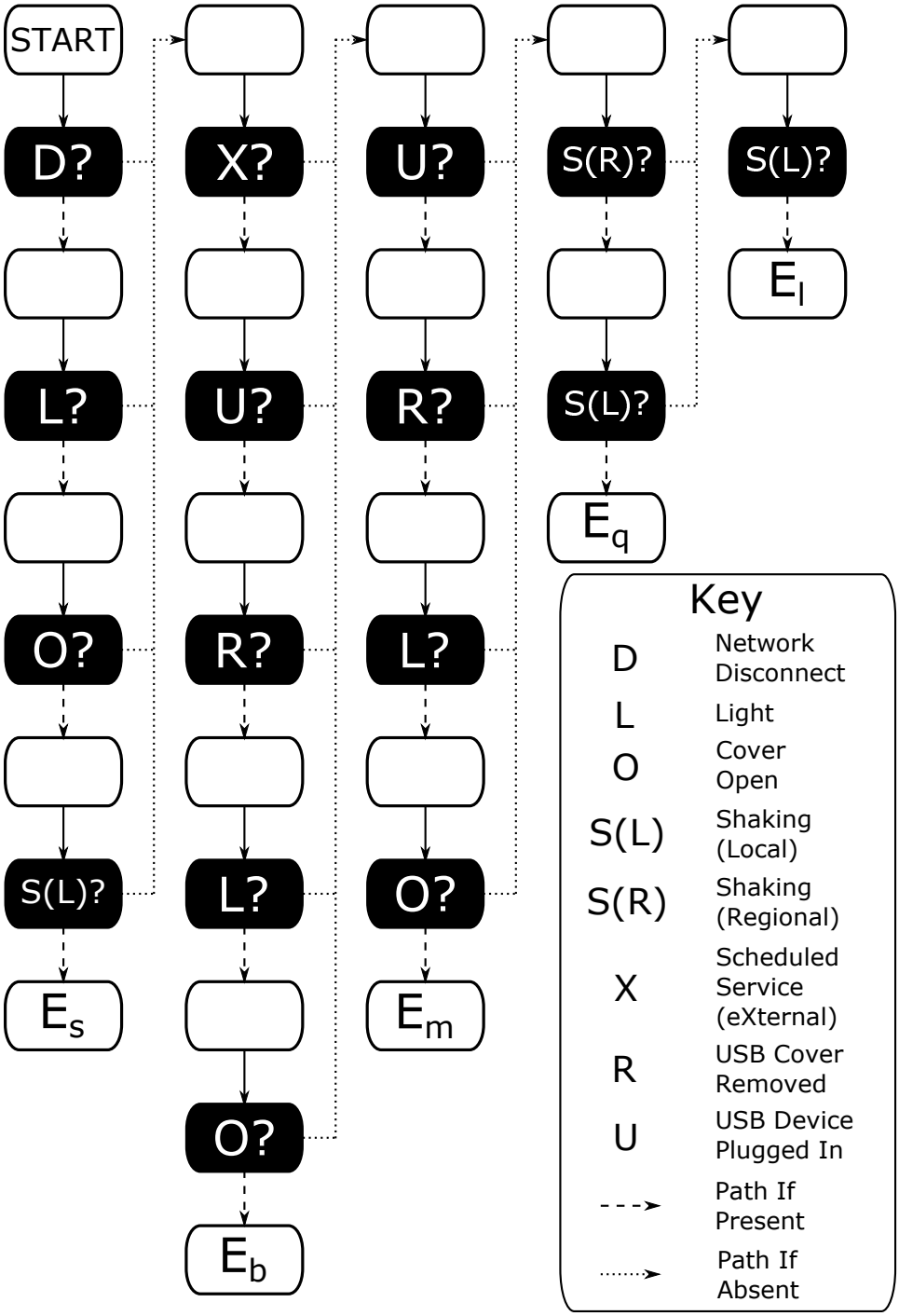


Figure 9.3: The full factor graph that TEDDI generates for the example in Section 9.4. Note that the limited factor graph would combine E_b and E_m, as well as E_q and E_l, because it would not be able to resolve external or global indicators and the events are otherwise identical.

and availability attributes, but in this case we would recommend slightly smaller goals such as “Edge device compromised” or “Edge device bypassed.”) These graphs, depending on the goals defined, could each wind up being as big as our full factor graph. TEDDI, in contrast, requires only a single graph be configured to cover the entire system.

- The RRE lacks an FGDSL-like system for simplifying the tree-building process; in fact, the authors explicitly state that the “ART model in RRE’s global server needs to be designed manually” [161]. This means that the operator must go through the long and time-consuming process of hand-crafting their own ARTs, and do it without the help of a tool like our generation tool.
- Finally, the presence of regional indicators means that tamper logic will have to be split across the local and global detection engines, an added hassle to operators who are already manually defining their own ARTs. TEDDI, on the other hand, relieves operators of this burden by building its own full and limited factor graphs.

Additionally, the RRE makes use of a Bayesian classifier to guard against the chance that an alert was sent in error, or the chance that an alert was not sent even though the something bad actually occurred. While this feature may help reduce potential false positives and negatives, the classifier uses probabilities that are “calculated based on historical information about the system” [161], further increasing the amount of data needed to allow the RRE to run.

Admittedly, RRE does offer some small advantages over TEDDI—for example, its ability to assign responses to each individual step within the tree eliminates the need for defining pre-events. On the whole, however, the drawbacks listed above make the RRE a bit more cumbersome to use than TEDDI.

SCADA-Hawk: As an anomaly-based protection system, SCADA-Hawk’s biggest issue

is that it models "normal" behavior when verifying system actions [123]. These models can be provided in two ways:

- The operator can manually define a sequence of events that feeds into SCADA-Hawk as an example of normal behavior. This, however, requires the operator to determine every possible normal behavior that can happen in the system, and manually construct an example for each behavior to give to the system. Defining such behaviors can be a tricky process, as it requires that the grid defender have complete knowledge of the tasks performed by every device in the network, and any behavior that is not defined may generate a false positive, potentially leading to increased costs and decreased system availability. In general, signature-based systems looking for bad behavior (such as TEDDI) have lower false positive rates than anomaly-based systems that verify normal behavior [68].
- The system can be placed into a learning mode, to record "snapshots" of normal behavior sequences. While this option is less time-consuming for the grid defender than the prior one, it means the operator has to delay deployment until SCADA-Hawk has seen enough system behavior before it can actually start protecting the network. The length of this learning period is at best arbitrary, as not allocating enough time to this task may mean legitimate but infrequent behaviors are missed, and will later appear as false positives.

In short, operators who attempt to deploy SCADA-Hawk are forced to spend a long period of time either training SCADA-Hawk or manually defining the system's expected behavior, all with the risk of high false positive rates if either method does not do enough to cover the system's behavior space. With TEDDI, on the other hand, a defender can select the events they are concerned about and get the system up and running more quickly, and usually with fewer false positives.

PQS: On the surface, PQS [105] has many of the same usability benefits as TEDDI:

- The process models can be constructed using a number of different methods, such as "state machines, formal language descriptions, Hidden Markov Models, kinematic descriptions, or a set of rules" [105]. The system even has a special markup language that these models are compiled into for submission to the system.
- The process models can be quickly constructed using the data found within a narrative. In the case of the Schweitzer scenario, the indicator sequence can quickly be parsed out and translated into one of the PQS-friendly models above.
- Increasing the amount/complexity of sequences does not significantly increase the complexity of the PQS model. Each event can be easily described in its own model.

The issue with PQS, however, is similar to that of the RRE: It does not provide any guidance as to how its models should be built—instead, it blindly uses the provided models without providing any feedback as to their quality. TEDDI, in comparison, goes a step farther by working with the user to build its sequences and using its Response Suggestion Engine to help the user refine their models. By building a model *with* TEDDI as opposed to *for* PQS, the user finishes with a more-functional model for their system.

PAC: The biggest issue with PAC [141] is that it relies on information that are not present in our given set of events. More specifically:

> "[PAC uses] *probabilistic methods for sensor correlation...Sensor coupling in this fashion is easily expressed in Bayes formalisms; specifically, the coupling is achieved by dynamically modifying priors in the TCP session monitor"* [141].

PAC also requires that users define alert similarities for specific scenarios and minimum values that similarity scores must reach before being reported, but these are analogous to the indicator sequences and monitor thresholds required by TEDDI.

This setup means that PAC requires some knowledge of the prior distribution regarding how sensors relate to one another, data that cannot be obtained just from our set of events. Much like SCADA-Hawk, this distribution data needs to be collected and analyzed beforehand, increasing the time and effort needed to put PAC into place.

SCPSE: SCPSE holds an initial configuration advantage over the RRE, as the attack graph templates (AGTs) used by SCPSE are generated automatically from the access control rules of the network [162]. However, the drawback from our PAC analysis also applies: To predict the attacker's path through the network "a posterior probability distribution over the AGT's state space is calculated according to the false positive and negative rates of the triggered and non-triggered IDS alerts, respectively" [162]. Again, such information is far beyond what the narrative can provide, and also far beyond what TEDDI requires to operate.

9.5 Summary

All together, our evaluation demonstrates that TEDDI answers all of our questions regarding its speed, accuracy, and configuration requirements, and performs comparably or better than current state-of-the-art protection solutions:

- We can process our factor graph fast enough to satisfy the constraints of the power grid. In our tests, TEDDI processed even our largest factor graph (99 function nodes) well within our 400 μs limit.
- Our event detection accuracy rates compare favorably to other protection systems. In our tests, TEDDI made the correct event decision (given 4 possible options) with

99.2% accuracy at the TIP level, and accurately computed the global state and made the proper event decision 100% of the time at the TDP level.

- Our TEDDI Generation Tool gives us a distinct usability edge over comparable systems, which either need more resources or more time than TEDDI to put together a protection plan.

These results suggest that TEDDI can operate effectively even under the constraints of the power grid, and can be configured to do so with considerably less hassle than other systems.

Chapter 10

Conclusions

In this thesis, we examined the need for securing edge devices installed in power grid SCADA networks, and highlighted the consequences of failing to secure them adequately. We introduced the *grid defender's dilemma*, a collection of conflicting interests that hinder grid defenders' efforts to secure the grid by making protection systems designed for standard IT networks infeasible to apply. We proposed a distributed, sensor-based method of tamper protection as an alternative, and introduced our TEDDI system as a prototype of such a method. We outlined the architecture of TEDDI, and described how its tamper information points, decision points, and enforcement points work together to make decisions and respond to the events it sees. We showed how TEDDI's improved data-gathering capabilities, flexible response strategy, and use of factor graphs [42] for data fusion allow it to work around all of the stumbling blocks that hinder prior work, and allow it to solve the grid defender's dilemma. We also proposed and developed the TEDDI Generation Tool, a program that lets a grid operator easily define and configure TEDDI for their own unique network, and outlined features such as FGDSL, the Response Suggestion Engine, the Network Topology Uploader, and the TDP Placement Tool, all of which enhance the program's usability. Finally, we evaluated TEDDI's performance and detection rate and analyzed its usability, and demonstrated that the program is faster, more accurate, and required less

effort to use than prior work.

In the future, we would like to relax some of the assumptions TEDDI makes about the network, as TEDDI's success is heavily reliant on the underlying network and sensors:

- TEDDI assumes that a TIP will always be able to reach its TDP, but what could it do if the connection were severed? Could it integrate itself into another TDP's dataset, or perhaps even work with other isolated TIPs to determine the regional state of the system? A fair amount of work already exists in the distributed-decision-making field (for example, Reidt, Srivatsa, and Balfe's distributed key-revocation proposal [103]), and some of these ideas could potentially be worked into TEDDI.

- TEDDI assumes that a monitor will always provide accurate readings, but what might happen if a sensor fails or is compromised? Bad data detection has been studied extensively in the fields of power system state estimation (for example, Niemira et al.'s analysis of how both real and reactive power measurements are affected by bad data injections [87]) and general sensor validation (such as Zhu et al.'s scheme for validating data from individual sensors operating within a sensor network [160]), and incorporating this work into TEDDI may make the system more resilient against crafty attackers.

- TEDDI assumes that attackers must compromise an edge device to reach a SCADA network, but what if an attacker ignores the device entirely and taps directly into the SCADA network, or reaches the network via a device's wireless access point? Handling this issue would likely require expanding TEDDI's monitoring scope to include network-specific indicators and look for malicious packets, much like Bro [21] and Snort [121].

  Additionally, including software or power-specific indicators in TEDDI's detection scheme would also be a useful nest step. Many protection systems, including TEDDI, limit themselves to looking at specific type of indicators, such as software signals

or physical sensors. By incorporating both types of data into a detection system, however, we could further improve our event detection capabilities by using one dataset to help confirm decisions based on the other, furthering our goal of accurate event detection.

Finally, we would like to get TEDDI into the hands of as many power professionals as possible, and evaluate TEDDI's speed, accuracy, and usability within a realistic power grid setting. While the industry professionals we demonstrated TEDDI for gave us very positive feedback, and we performed our own analyses that showed how TEDDI bested prior work in this front, we were not able to have grid defenders use TEDDI directly within their own grid environments. Feedback from such hands-on tests would be immensely valuable, and would help us refine and enhance TEDDI's features and make the system even more useful for grid defenders.

We hope that TEDDI inspires others to take a harder look at how to secure SCADA networks operating in the power grid. These networks have fundamentally different goals and challenges than traditional IT networks, and simply taking a system that is designed for a "normal" network and sticking it into the power grid may cause more harm than good. However, we can apply *some* existing solutions to the grid defender's dilemma, and properly applying just the right ideas can strike a workable balance between securing grid networks and respecting their availability needs. With TEDDI, we show that striking such a balance is possible, and take the first step towards finally securing some of our nation's most critical infrastructure.

Bibliography

[1] ABB. Why use reclosers?, 2016. http://www.abb.com/product/ap/db0003db004279/B0B2C0094A20CB88C1257A0E004C685A.aspx.

[2] Ali Abbasi, Jos Wetzels, Wouter Bokslag, Emmanuele Zambon, and Sandro Etalle. On emulation-based network intrusion detection systems. In *Research in Attacks, Intrusions, and Defenses (RAID) Symposium*, 2014.

[3] Marshall Abrams and Joe Weiss. Malicious control system cyber security attack case study - Maroochy water services, Australia, 2008. http://csrc.nist.gov/groups/SMA/fisma/ics/documents/Maroochy-Water-Services-Case-Study_report.pdf.

[4] Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6), June 1978.

[5] AlertLogic ActiveWatch Premier. http://www.alertlogic.com/products-services/activewatch/activewatch-premier/.

[6] Muhammad Qasim Ali and Ehab Al-Shaer. Configuration-based IDS for advanced metering infrastructure. In *The 20th ACM Conference on Computer and Communications Security (ACM CCS)*, 2013.

[7] Bernhard Amann, Robin Sommer, Aashish Sharma, and Seth Hall. A lone wolf no more: Supporting network intrusion detection with real-time intelligence. In

The 15th International Symposium on Resarch in Attacks, Intrusions, and Defenses, 2012.

[8] Bertrand Anckaert, Bjorn De Sutter, and Koen De Bosschere. Software piracy prevention through diversity. In *Proceedings of the 4th ACM Workshop on Digital Rights Management*, 2004.

[9] Atmel Corporation. Atmel Trusted Platform Module, 2015. `http://www.atmel.com/products/security-ics/embedded/default.aspx`.

[10] Ahmed M. Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Butler, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. Hypervision across worlds: Real-time kernel protection from the ARM TrustZone secure world. In *The 21st ACM Conference on Computer and Communications Security (ACM CCS)*, 2014.

[11] Hajar Benmoussa, Anas Abou El Kalam, and Abdallah Ait Ouahman. Distributed intrusion detection system based on anticipation and prediction approach. In *The 12th International Conference on Security and Cryptography (SECRYPT)*, 2015.

[12] Alberto Berizzi. The Italian 2003 blackout. In *The IEEE Power Engineering Society General Meeting*, 2004.

[13] Robin Berthier and William H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *Proceedings of the 17th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2011.

[14] Robin Berthier and William H. Sanders. Monitoring advanced metering infrastructures with Amilyzer. In *Cybersecurity of SCADA and Industrial Control Systems*, 2013.

[15] Robin Berthier and William H. Sanders. Monitoring advanced metering infrastructures with Amilyzer. In *Proceedings of the Cybersecurity of SCADA and Industrial Control Systems*, 2013.

[16] Swarup Bhunia, Miron Abramovici, Dakshi Agarwal, Paul Bradley, Michael S. Hsiao, Jim Plusquellic, and Mohammad Tehranipoor. Protection against hardware trojan attacks: Towards a comprehensive solution. *IEEE Design & Test*, 30(3):6–17, 2013.

[17] Nathaniel Boggs, Sharath Hiremagalore, Angelos Stavrou, and Salvatore J. Stolfo. Cross-domain collaborative anomaly detection: So far yet so close. In *The 14th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2011.

[18] Atul Bohara, Uttam Thakore, and William H. Sanders. Intrusion detection in enterprise systems by combining and clustering diverse monitor data. In *Symposium and Bootcamp on the Science of Security*, 2016.

[19] Leonid Bolotnyy and Gabriel Robins. Physically unclonable function-based security and privacy in RFID systems. In *The Fifth Annual IEEE International Conference on Pervasive Computing and Communications*, 2007.

[20] Marshall Brain. How power grids work, 2004. `http://www.science.smith.edu/~jcardell/Courses/EGR220/ElecPwr_HSW.html`.

[21] The Bro network security monitor. `https://www.bro.org/`.

[22] Helena Cain. What is a roman signet ring? eHow.com. `http://www.ehow.com/about_6615340_roman-signet-ring_.html`.

[23] Phuong Cao, Eric Badger, Zbigniew Kalbarczyk, Ravishankar Iyer, and Adam Slagell. Preemptive intrusion detection: Theoretical framework and real-world measurements. In *Symposium and Bootcamp on the Science of Security*, 2015.

[24] Satish Chandra, Bradley Richards, and James R. Larus. Teapot: A domain-specific language for writing cache coherence protocols. *IEEE Transactions On Software Engineering*, 25(3), May/June 1999.

[25] Hoi Chang and Mikhail J. Atallah. Protecting software codes by guards. Technical Report 2001-49, The Center for Education and Research in Information Assurance and Security, Purdue University, 2001.

[26] Senthilkumar G. Cheetancheri, John Mark Agosta, Denver H. Dash, Karl N. Levitt, Jeff Rowe, and Eve M. Schooler. A distributed host-based worm detection system. In *The 2006 SIGCOMM Workshop on Large-Scale Attack Defense (LSAD)*, 2006.

[27] Steven Cheung, Bruno Dutretre, Martin Fong, Ulf Lindqvist, Keith Skinner, and Alfonso Valdes. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA Security Scientific Symposium*. Springer, 2007.

[28] Richard Clayton and Mike Bond. Experience using a low-cost FPGA design to crack DES keys. In *Workshop on Cryptographic Hardware and Embedded Systems*, 2002.

[29] Christian Collberg, Sam Martin, Jonathan Myers, and Jasvir Nagra. Distributed application tamper detection via continuous software updates. In *The Annual Computer Security Applications Conference (ACSAC)*, 2012.

[30] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction To Algorithms*. The MIT Press, 3rd edition, 2009.

[31] Jason Crampton. XACML and role-based access control. Lecture from the DIMACS Workshop on Security of Web Services and E-Commerce, 2005.

[32] Jordi Cucurull, Mikael Asplund, and Simin Nadjm-Tehrani. Anomaly detection and mitigation for disaster area networks. In *The 13th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2010.

[33] Rubens Alexandre de Faria, Keiko V. Ono Fonseca, Bertoldo Schneider Jr., and Sing Kiong Nguang. Collusion and fraud detection on electronic energy meters: A use case of forensics investigation procedures. In *IEEE Security and Privacy Workshops (SPW)*, 2014.

[34] Dell Precision 340. `http://www.dell.com/support/home/us/en/19/product-support/product/precision-340/configuration`.

[35] Soma Shekara Sreenadh Reddy Depuru, Lingfeng Wang, Vijay Devabhaktuni, and Nikhil Gudi. Smart meters for power grid–challenges, issues, advantages and status. In *Power Systems Conference and Exposition (PSCE)*, 2011.

[36] Avinash Desai. Anti-counterfeit and anti-tamper implementation using hardware obfuscation. Master's thesis, Virginia Polytechnic Institute and State University, August 2013.

[37] Silvio Dragone. Physical security protection based on non-deterministic configuration of integrated microelectronic security features. In *The First International Cryptographic Module Conference*, September 2013.

[38] Patrick Düssel, Christian Gehl, Pavel Laskov, Jens-Uwe Bußer, Christof Störmann, and Jan Kästner. Cyber-critical infrastructure protection using real-time payload-based anomaly detection. In Erich Rome and Robin Bloomfield, editors, *Critical Information Infrastructures Security*, volume 6027 of *Lecture Notes In Computer Science*, pages 85–97. Springer Berlin Heidelberg, 2010.

[39] Nathan J. Edwards. Hardware intrusion detection for supply-chain threats to critical infrastructure embedded systems. Master's thesis, University of Illinois at Urbana-Champaign, 2012.

[40] Nicolas Falliere, Liam O. Murchu, and Eric Chien. *W32.Stuxnet Dossier*. Symantec Corporation, February 2011. `https://www.symantec.com/content/en/`

us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[41] Margus Freudenthal. Domain specific languages in a customs information system. *IEEE Software*, PP(99), 2009.

[42] Brendan Frey. Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, 2003.

[43] J. T. Giffin, M. Christodorescu, and L. Kruger. Strengthening software self-checksumming via self-modifying code. In *The 21st Annual Computer Security Applications Conference*, 2005.

[44] Global Energy Partners. *OG&E Smart Study Together Impact Results*, February 2012. C. Williamson and J. Shishido, Principal Investgators.

[45] Niv Goldenburg and Avishai Wool. Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75, 2013.

[46] Leonard L. Grigsby, editor. *Electric Power Generation, Transmission, and Distribution*. CRC Press, 2007.

[47] David Grochocki, Jun Ho Huh, Robin Berthier, Rakesh Bobba, William H. Sanders, Alvaro A. Cárdenas, and Jorjeta G. Jetcheva. AMI threats, intrusion detection requirements and deployment recommendations. In *Proceedings of the 3rd IEEE International Conference on Smart Grid Communications*, 2012.

[48] Adam Hahn and Manimaran Govindarasu. Model-based intrusion detection for the smart grid (MINDS). In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW)*, 2013.

[49] Sinclair Hansen. An intrusion detection system for supervisory control and data acquisition systems. Master's thesis, Queensland University of Technology, March 2008.

[50] John Harrison. *Blackout of 1996*. The Northwest Power and Conservation Council, October 2008. https://www.nwcouncil.org/history/Blackout.

[51] Hewlett-Packard. HP trusted platform module, 2013. http://h18004.www1.hp.com/products/servers/proliantstorage/module.html.

[52] Dorit S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(1):148–162, December 1982.

[53] HowStuffWorks. What are amps, watts, volts and ohms?: Electrical efficiency, 2000. http://science.howstuffworks.com/environmental/energy/question5011.htm.

[54] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In *The 6th International Conference on Information-Warfare & Security*, 2011.

[55] IBM. IBM 4765 PCIe Data Sheet, 2011. http://www-03.ibm.com/security/cryptocards/pciecc/pdf/PCIe_Spec_Sheet.pdf.

[56] IEEE. IEEE standard communication delivery time performance requirements for electric power substation automation. IEEE Standard 1646-2004, 2005. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1405811.

[57] Vadim Indelman, Stephen Williams, Michael Kaess, and Frank Dellaert. Information fusion in navigation systems via factor graph based incremental smoothing. *Robotics and Autonomous Systems*, 61(8):721–738, 2013.

[58] Intel Corporation. Intel trusted platform module (TPM-AXXTPME3/AXXTPME5) hardware user's guide, 2011. `http://download.intel.com/support/motherboards/server/sb/g21682004_tpm_hwug1.pdf`.

[59] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through VMM-based "out-of-the-box" semantic view reconstruction. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[60] Yier Jin and Yiorgos Makris. Hardware trojan detection using path delay fingerprint. In *IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008.

[61] Paria Jokar, Hasen Nicanfar, and Victor C.M. Leung. Specification-based intrusion detection for home area networks in smart grids. In *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2011.

[62] Alfred C. Herndon Jr., Joseph L. Ehasz, and Kermit Paul. *Taum Sauk Upper Dam Breach*. Federal Energy Regulatory Commission, May 2006. FERC No. P-2277. `http://www.ferc.gov/industries/hydropower/safety/projects/taum-sauk/ipoc-rpt/full-rpt.pdf`.

[63] George G. Karady. Concept of energy transmission and distribution. In Leonard L. Grigsby, editor, *Electric Power Generation, Transmission, and Distribution*, Electric Power Engineering Handbook, pages 8–1—8–12. CRC Press, second edition, 2007.

[64] Tayeb Kenaza, Abdenour Labed, Yacine Boulahia, and Mohcen Sebehi. Adaptive SVDD-based learning for false alarm reduction in intrusion detection. In *The 12th International Conference on Security and Cryptography (SECRYPT)*, 2015.

[65] Stephen Kent. *Protecting Externally Supplied Software in Small Computers*. PhD thesis, Massachusetts Institute of Technology, September 1980.

[66] Ross Kindermann and James Laurie Snell. *Markov Random Fields and their Applications*. American Mathematical Society, 1980.

[67] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication (RFC 2104)*. Internet Engineering Task Force, February 1997. `https://tools.ietf.org/html/rfc2104`.

[68] Manish Kumar, M. Hanumanthappa, and T. V. Suresh Kumar. Intrusion detection system - false positive alert reduction technique. *ACEEE International Journal on Network Security*, 2(3):37–40, July 2011.

[69] Aron Laszka, Waseem Abbas, S. Shankar Sastry, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. Optimal thresholds for intrusion detection systems. In *Symposium and Bootcamp on the Science of Security*, 2016.

[70] Robert M. Lee, Michael J. Assante, and Tim Conway. *Analysis of the Cyber Attack on the Ukranian Power Grid*. Joint work between SANS ICS and the Electricity Information Sharing and Analysis Center, March 2016.

[71] Hui Lin, Homa Alemzadeh, Daniel Chen, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Safety-critical cyber-physical attacks: Analysis, detection, and mitigation. In *Symposium and Bootcamp on the Science of Security*, 2016.

[72] Lionel Litty, H. Andrés Lagar-Cavilla, and David Lie. Hypervisor support for identifying covertly executing binaries. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[73] Hans-Andrea Loeliger, Justin Dauwels, Junli Hu, Sascha Korl, Li Ping, and Frank R. Kschischang. The factor graph approach to model-based signal processing. *Proceedings of the IEEE*, 95(6):1295–1322, 2007.

[74] Steve Lusk, Alex Amirnovin, and Tim Collins. Cyber-intrusion auto-response and policy management system (CAPMS). Presentation at the Cybersecurity for Energy Delivery Systems Peer Review, August 2014. `https://www.controlsystemsroadmap.net/ieRoadmap%20Documents/ViaSat-CAPMS-CEDS_Peer_Review_2014.pdf`.

[75] Abhranil Maiti and Patrick Schaumont. Improving the quality of a physical unclonable function using configurable ring oscillators. In *International Conference on Field Programmable Logic and Applications*, 2009.

[76] Ananth Mavinakayanahalli, Prasanna Panchamukhi, Jim Keniston, Anil Keshavamurthy, and Masami Hiramatsu. Probing the guts of Kprobes. In *Proceedings of the Ottawa Linux Symposium (OLS)*, 2006.

[77] McAfee cloud-based security for SMBs. `http://www.mcafee.com/us/resources/data-sheets/ds-cloud-based-security-for-smbs.pdf`.

[78] Rajesh Kannan Megalingam, Ashok Krishnan, Bharath Kalathiparambil Ranjan, and Amar Kelu Nair. Advanced digital smart meter for dynamic billing, tamper detection, and consumer awareness. In *Proceedings of the 3rd International Conference on Electronics Computer Technology*, 2011.

[79] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *The 14th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2011.

[80] Ameren Missouri. Taum sauk energy center, 2016. https://www.ameren.com/missouri/environment/hydroelectric/taum-sauk-information.

[81] Robert Mitchell and Ing-Ray Chen. Behavior-rule based intrusion detection systems for safety critical smart grid applications. *IEEE Transactions on Smart Grid*, 4(3):1254–1263, September 2013.

[82] Motorola. ACE 3600 specifications sheet, 2009. http://www.motorolasolutions.com/web/Business/Products/SCADA%20Products/ACE3600/_Documents/Static%20Files/ACE3600%20Specifications%20Sheet.pdf.

[83] National Institute of Standards and Technology. Validated FIPS 140-1 and fips 140-2 cryptographic modules, 2016. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm.

[84] North American Electric Reliability Corporation (NERC). CIP compliance, 2016. http://www.nerc.com/pa/CI/Comp/Pages/default.aspx.

[85] Samuel Neves and Filipe Araujo. Binary code obfuscation through C++ template metaprogramming. In *INForum Simpósio de Informática*, 2012.

[86] NextEra Energy Resources. Solar - how solar plants work, 2016. http://www.nexteraenergyresources.com/what/solar_works.shtml.

[87] William Niemira, Rakesh B. Bobba, Peter Sauer, and William H. Sanders. Malicious data detection in state estimation leveraging system losses & estimation of perturbed parameters. In *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2013.

[88] Ben Niewenhuis, R. D. (Shawn) Blanton, Mudit Bhargava, and Ken Mai. SCAN-PUF: A low overhead physically unclonable function from scan chain power-up states. In *IEEE International Test Conference (ITC)*, 2013.

[89] OASIS cover pages: Extensible access control markup language (XACML), 2009. `http://xml.coverpages.org/xacml.html`.

[90] Hamed Okhravi, James Riordan, and Kevin Carter. Quantitative evaluation of dynamic platform techniques as a defense mechanism. In *Research in Attacks, Intrusions, and Defenses (RAID) Symposium*, 2014.

[91] Organization for the Advancement of Structured Information Standards. *eXtensible Access Control Markup Language (XACML) Version 3.0*, January 2013. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf`.

[92] Andrea Peterson. *Are squirrels a bigger threat to the power grid than hackers?* Washington Post, January 2016. `https://www.washingtonpost.com/news/the-switch/wp/2016/01/12/are-squirrels-a-bigger-threat-to-the-power-grid-than-hackers/`.

[93] Nick L. Petroni, Jr. and Michael Hicks. Automated detection of persistent kernel control-flow attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[94] John R. Pierce. *An Introduction to Information Theory: Symbols, Signals and Noise*. Dover Publications, Inc., 2nd, revised edition, 1980.

[95] Miodrag Potkonjak, Ani Nahapetian, Michael Nelson, and Tammara Massey. Hardware trojan horse detection using gate-level characterization. In *46th ACM/IEEE Design Automation Conference*, 2009.

[96] Thomas H. Ptacek and Timothy N. Newsham. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Secure Networks, Inc., January 1998. `http://insecure.org/stf/secnet_ids/secnet_ids.html`.

[97] Ashwin Ramaswamy. Autoscopy: Detecting pattern-searching rootkits via control flow tracing. Master's thesis, Dartmouth College, May 2009.

[98] Raspberry Pi 2 Model B. `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`.

[99] Jason Reeves. Autoscopy Jr.: Intrusion detection for embedded control systems. Master's thesis, Dartmouth College, September 2011. Revised version of August 2011 thesis submission.

[100] Jason Reeves, Ashwin Ramaswamy, Michael Locasto, Sergey Bratus, and Sean W. Smith. Intrusion detection for resource-constrained embedded control systems in the power grid. *International Journal of Critical Infrastructure Protection*, 5(2):74–83, 2012.

[101] Jason Reeves and Sean W. Smith. Tamper event detection on distributed devices in critical infrastructure. In *The Second International Cryptographic Module Conference (ICMC)*, 2014.

[102] Jason Reeves and Sean W. Smith. Solving the grid defender's dilemma: Tamper protection for distributed cyber-physical systems. In *The 12th International Conference on Security and Cryptography (SECRYPT)*, 2015.

[103] Steffen Reidt, Mudhakar Srivatsa, and Shane Balfe. The fable of the bees: Incentivizing robust revocation decision making in ad hoc networks. In *The 16th ACM Conference on Computer and Communications Security (ACM CCS)*, 2009.

[104] Ryan Riley, Xuxian Jiang, and Dongyan Xu. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing. In *The 11th International Symposium on Recent Advances in Intrusion Detection*, 2008.

[105] Christopher Roblee, Vincent Berk, and George Cybenko. Large-scale autonomic server monitoring using process query systems. In *IEEE International Conference on Autonomic Computing*, 2005.

[106] Tanya Roosta, Dennis K. Nilsson, Ulf Lindqvist, and Alfonso Valdes. An intrusion detection system for wireless process control systems. In *The 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2008.

[107] Ulrich Rührmair and Marten van Dijk. PUFs in security protocols: Attack models and security evaluations. In *IEEE Symposium on Security and Privacy*, 2013.

[108] Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. A novel technique for improving hardware trojan detection and reducing trojan activation time. *IEEE Transactions on Very Large Scale Integration Systems*, 20(1):112–125, January 2012.

[109] Josh Schellenberg. Evaluating the total cost of outages. Presentation to the Distribution Reliability Working Group at the IEEE Power & Energy Society General Meeting, July 2012. `http://grouper.ieee.org/groups/td/dist/sd/doc/2012-07-04-Evaluating-the-Total-Cost-of-Outages.pdf`.

[110] Schweitzer Engineering Laboratories. SEL-651R-2 Recloser Control Data Sheet, 2013. `https://www.selinc.com/WorkArea/DownloadAsset.aspx?id=100135`.

[111] SEL-3355 computer data sheet. https://cdn.selinc.com/assets/Literature/Product%20Literature/Data%20Sheets/3355_DS_20160105.pdf.

[112] SEL-3622 security gateway data sheet. https://cdn.selinc.com/assets/Literature/Product%20Literature/Data%20Sheets/3622_DS_20151230.pdf.

[113] SEL-651R advanced recloser control. https://www.selinc.com/SEL-651R/.

[114] Sumit Siddharth. *Evading NIDS, revisited*. Symantec Corporation, November 2010. http://www.symantec.com/connect/articles/evading-nids-revisited.

[115] Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition, 2007.

[116] Rebecca Smith. U.S. risks national blackout from small-scale attack. *Wall Street Journal*, March 2014. http://online.wsj.com/news/articles/SB10001424052702304020104579433670284061220.

[117] Sean Smith and John Marchesini. *The Craft of System Security*. Addison-Wesley Professional, 2007.

[118] Sean W. Smith. *Trusted Computing Platforms: Design and Applications*. Springer, 2005.

[119] Sean W. Smith, Elaine Palmer, and Steve Weingart. Using a high-performance, programmable secure coprocessor. In *Second International Conference on Financial Cryptography*, 1998.

[120] Sean W. Smith and Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31(1999):831–860, 1999.

[121] Snort. `http://www.snort.org/`.

[122] Rouslan V. Solomakhin. Predictive YASIR: High security with lower latency in legacy SCADA. Master's thesis, Dartmouth College, June 2010.

[123] William L. Sousan, Quiming Zhu, Robin Gandhi, and William Mahoney. Smart grid tamper detection using learned event patterns. In Vijay Pappu, Marco Carvalho, and Panos Pardalos, editors, *Optimization and Security Challenges in Smart Power Grids*, Energy Systems, pages 99–115. Springer Berlin Heidelberg, 2013.

[124] Diomidis Spinellis. Notable design patterns for domain-specific languages. *Journal of Systems and Software*, 56(1):91–99, February 2001.

[125] G. Edward Suh, Charles W. O'Donnell, and Srinivas Devadas. AEGIS: A single-chip secure processor. *Information Security Technical Report*, 10(2):63–73, 2005.

[126] TekTrakker Information Systems. Smart grid RFI: Addressing policy and logistical challenges. Comments on the DOE Request for Information of the same name, 2010. `http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/TekTrakker_Comments.pdf`.

[127] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. Unsupervised anomaly-based malware detection using hardware features. In *Research in Attacks, Intrusions, and Defenses (RAID) Symposium*, 2014.

[128] George Theodorakopoulos and John S. Baras. On trust models and trust evaluation metrics for ad hoc networks. *IEEE Journal on Selected Areas In Communications*, 24(2):318–328, 2006.

[129] Tyler Thia. *Signature-based detection, protection systems ineffective*. ZDNet, June 2011. http://www.zdnet.com/signature-based-detection-protection-systems-ineffective-2062300935/.

[130] Scott A. Thibault, Renaud Marlet, and Charles Consel. Domain-specific languages: From design to implementation application to video device drivers generation. *IEEE Transactions On Software Engineering*, 25(3), May/June 1999.

[131] Hing-Chung Tsang, Moon-Chuen Lee, and Chi-Man Pun. A robust anti-tamper protection scheme. In *Sixth International Conference on Availability, Reliability and Security (ARES)*, 2011.

[132] Katherine Tweed. *Hack Your Meter While You Can*. Greentech Media, April 2010. http://www.greentechmedia.com/articles/read/hack-your-meter-while-you-can.

[133] J. Doug Tygar and Bennet Yee. Dyad: A system for using physically secure coprocessors. In *Technological Strategies for the Protection of Intellectual Property in the Networked Multimedia Environment*, 1994.

[134] United States. *Federal Information Processing Standards Publication 140-1: Security Requirements for Cryptographic Modules*. National Institute for Standards and Technology, January 1994. http://csrc.nist.gov/publications/fips/fips1401.htm.

[135] United States. *Federal Information Processing Standards Publication 140-2: Security Requirements for Cryptographic Modules*. National Institute for Standards and Technology, May 2001. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[136] United States. *Federal Information Processing Standards Publication 140-3 (DRAFT): Security Requirements for Cryptographic Modules*. National Institute

for Standards and Technology, 2007. http://csrc.nist.gov/groups/ST/FIPS140_3/documents/FIPS_140-3%20Final_Draft_2007.pdf.

[137] United States. *ShakeMap Scientific Background*. United States Geological Survey, March 2011. http://earthquake.usgs.gov/earthquakes/shakemap/background.php.

[138] United States. *Cyber-Intrusion Auto-Response Policy and Management System (CAPMS)*. Department of Energy, May 2015. https://www.controlsystemsroadmap.net/ieRoadmap%20Documents/CAPMS_flyer.pdf.

[139] United States. *What is the electric power grid and what are some challenges it faces?* United States Energy Information Administration, December 2015. http://www.eia.gov/energy_in_brief/article/power_grid.cfm.

[140] UPX: The Ultimate Packer for eXecutables. http://upx.sourceforge.net/.

[141] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *The 4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2001.

[142] Joel Van Der Woude. Dependable cyber-physical systems through control flow monitoring. Undergraduate senior thesis, University of Illinois at Urbana-Champaign, May 2013.

[143] Elizabeth Von Meier. *Electric Power Systems: A Conceptual Introduction*. John Wiley and Sons, Inc., 2006.

[144] Matthew L. Wald. For the smart grid, a 'Synchrophasor'. NYTimes.com. http://green.blogs.nytimes.com/2010/04/01/for-the-smart-grid-a-synchophasor/.

[145] Yong Wang, Zhaoyan Xu, Jialong Zhang, Lei Xu, Haopei Wang, and Guofei Gu. SRID: State relation based intrusion detection for false data injection attacks in SCADA. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security – ESORICS 2014*, volume 8713 of *Lecture Notes in Computer Science*, pages 401–418. Springer International Publishing, 2014.

[146] Yujue Wang and Carl Hauser. An evidence-based Bayesian trust assessment framework for critical-infrastructure decision processing. In *The Fifth Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection*, 2011.

[147] Zhi Wang, Xuxian Jiang, Weidong Cui, and Peng Ning. Countering kernel rootkits with lightweight hook protection. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[148] Tucker Ward. Grid cryptographic simulation: A simulator to evaluate the scalability of the X.509 standard in the smart grid. Undergraduate senior thesis, Dartmouth College, 2013.

[149] Gabriel Weaver, Carmen Cheh, Edmund Rogers, William H. Sanders, and Dennis Gammel. Toward a cyber-physical topology language: Applications to NERC CIP audit. In *ACM Workshop on Smart Energy Grid Security (SEGS)*, 2013.

[150] Webroot SecureAnywhere business endpoint protection. `http://www.webroot.com/us/en/business/products/endpoint/`.

[151] Steve Weingart. Physical security devices for computer subsystems: A survey of attacks and defenses 2008 (updated from the CHES 2000 version), 2008. Originally from *Second International Workshop on Cryptographic Hardware and Embedded Systems*, August 2000.

[152] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. *Terminology for Policy-Based Management (RFC 3198)*. Internet Engineering Task Force, November 2001. `http://www.ietf.org/rfc/rfc3198`.

[153] Steve White and Liam Comerford. ABYSS: A trusted architecture for software protection. In *IEEE Symposium on Security and Privacy*, 1987.

[154] Steve White, Steve H. Weingart, William Arnold, and Elaine Palmer. Introduction to the Citadel architecture: Security in physically exposed environments. Technical Report RC16672, IBM T. J. Watson Research Center, 1991.

[155] WorldStandards.eu. Plug, socket & voltage by country, 2016. `http://www.worldstandards.eu/electricity/plug-voltage-by-country/`.

[156] Murty V. V. S. Yalla. A digital multifunction protection relay. *IEEE Transactions on Power Delivery*, 7(1):193–201, 1992.

[157] Yi Yang, Kieran McLaughlin, Timothy Littler, Sakir Sezer, and Haifeng Wang. Rule-based intrusion detection system for SCADA networks. In *Proceedings of the 2nd IET International Conference in Renewable Power Generation (RPG)*, 2013.

[158] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Jung-Eun Kim, and Lui Sha. Securecore: A multicore-based intrusion detection architecture for real-time embedded systems. In *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013.

[159] Jun Zhao, Qian He, and Linlin Yao. A distributed website anti-tamper system based on filter driver and proxy. In David Jin and Sally Lin, editors, *Advances in Multimedia, Software Engineering and Computing Vol.1*, volume 128 of *Advances in Intelligent and Soft Computing*, pages 415–421. Springer Berlin Heidelberg, 2012.

[160] Sencun Zhu, Sanjeev Setia, Sushil Jajodia, and Peng Ning. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *IEEE Symposium on Security and Privacy*, 2004.

[161] Saman Zonouz, Himanshu Khurana, William H. Sanders, and Timothy Yardley. RRE: A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):395–406, 2014.

[162] Saman Zonouz, Katherine Rogers, Robin Berthier, Rakesh Bobba, William H. Sanders, and Thomas Overbye. SCPSE: Security-oriented cyber-physical state estimation for power grid critical infrastructures. *IEEE Transactions on Smart Grid*, 3(4):1790–1799, 2012.