Specification-based Intrusion Detection for Advanced Metering Infrastructures

Robin Berthier and William H. Sanders
Coordinated Science Laboratory, Information Trust Institute, and
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
rgb@illinois.edu and whs@illinois.edu

Abstract—**It is critical to develop an effective way to monitor advanced metering infrastructures (AMI). To ensure the security and reliability of a modernized power grid, the current deployment of millions of smart meters requires the development of innovative situational awareness solutions to prevent compromised devices from impacting the stability of the grid and the reliability of the energy distribution infrastructure. To address this issue, we introduce a specification-based intrusion detection sensor that can be deployed in the field to identify security threats in real time. This sensor monitors the traffic among meters and access points at the network, transport, and application layers to ensure that devices are running in a secure state and their operations respect a specified security policy. It does this by implementing a set of constraints on transmissions made using the C12.22 standard protocol that ensure that all violations of the specified security policy will be detected. The soundness of these constraints was verified using a formal framework, and a prototype implementation of the sensor was evaluated with realistic AMI network traffic.**

I. Introduction

The current worldwide upgrade of the power grid into a *smart grid* that leverages modern communication infrastructure to increase reliability and efficiency is also creating important security challenges. The protection of the many components of the grid against cyber-threats has always been critical, but the recent discovery of real and complex attacks [1] has propelled the concern to a new level.

The objectives of this paper are to study and address the specific issue of intrusion detection in *advanced metering infrastructures* (AMI). An AMI is a communication infrastructure that enables meters and utilities to exchange information such as power consumption, price update, or outage awareness. Smart meters play the key role of gateway between the home area network (HAN) at the customer's premises and the utility network. Their functionality make them an interesting target for attackers [2], and the impact of a virus that could propagate among meters and issue remote disconnect commands to a large set of devices could be catastrophic. The vulnerabilities found in several meter implementations [3] over the past few years further motivate the importance of the design and deployment of a highly reliable monitoring solution for AMIs.

It is important to understand the specific properties and constraints of this unique infrastructure in order to develop a relevant intrusion detection system (IDS). We argue that traditional IT solutions that rely on standard misuse-based or anomaly-based detection technologies are not suitable because they suffer from shortcomings that are incompatible with the limitations and monitoring requirements of AMIs. For example, the lack of information about existing attacks in AMIs and the need to be able to detect unknown attacks make it impossible to use signature-based solutions. On the other hand, the need to rapidly understand the root causes of attacks and the need to detect stealthy threats limit the use of anomaly-based solutions.

As a result, we introduce a specification-based intrusion detection sensor to accurately monitor the traffic at the edge of an AMI network. Reliance on specifications rather than attack signatures or statistical profiles enables our solution to detect unknown threats while providing detailed information about malicious behaviors detected. Specification-based detection technologies are known for having two important limitations. First, the development of the specifications is an expensive and tedious process. Second, specifications are often very difficult to evaluate and verify. The tight control over communication protocols authorized in AMIs and the homogeneous behavior of meters and metering traffic greatly reduce the impact of the first issue. We address the second, and more important, issue by leveraging a formal verification framework [4] that we combine with empirical evaluations. Specifically, our contributions are the following.

1) We define a set of *security specifications* for smart meters and a *security policy* for AMI to prevent compromised meters from impacting the infrastructure;
2) We develop a *formal model for the C12.22 standard protocol*, which is the application layer protocol used by meters to exchange data and receive configuration information;
3) We conduct a *formal verification of the specifications and monitoring operations* at the application-layer in order to guarantee that no attack can violate the security policy without being detected; and
4) We describe a *prototype implementation* that we evaluate in an emulated but realistic AMI environment.

This paper is organized as follows. We review the related work in Section II, and we present our approach, along with background information about AMI and the threat model in Section III. We present standard protocols for AMI and related security requirements in Section IV. The formal verification of these specifications is reviewed in Section V. Finally, we detail our prototype implementation in Section VI and empirical evaluation in Section VII. We conclude and discuss future work in Section VIII.

II. Related Work

This section is dedicated to review of related publications. In the first part, we present the field of specification-based intrusion detection, and in the second part, we review the literature on AMI security.

A. Specification-based Intrusion Detection

The concept of specification-based intrusion detection was first introduced by C. Ko in 1996 [5]. It describes the desirable behavior of a system through its functionalities and through the security policy. Any sequence of operations executed outside of the system's specifications is considered to be a security violation. Various formalisms have been proposed to capture valid operation sequences of systems, either manually with the parallel environment grammar [5], regular expressions for events [6], [7], or abstract state machine language [8], or automatically with techniques such as inductive logic programming [9], software fault tree, and colored Petri nets [10]. Specification-based approaches have also been integrated with automated response [11] or combined with anomaly-based techniques [12], [13] to keep high detection accuracy while decreasing the cost of manually defining the specifications.

The cost of defining the specifications has always remained a barrier to the extension of the application of specification-based intrusion detection beyond a small set of protocols. So far, security specifications have been studied and defined for routing protocols such as AODV [14], [15], [16], or OSLR [17], Voice over IP protocols [18], [19], [20], carrier Ethernets [21], and a wireless network infrastructure [22].

Another limitation of specification-based approaches has been the difficulty of verifying that the specifications are correct and that they cover the threat model. Important efforts have been made by Song et al. to address this issue by applying a formal verification framework [23], [4], [24]. Additional background information about this framework is provided in the next section, and we describe in Section V how we leverage this framework to verify the monitoring operations for the C12.22 standard protocol.

B. AMI Security

The security requirements for AMIs, and related threats, are discussed in [25]; the author emphasizes the fact that encryption and authentication alone will not be sufficient security protections. A comprehensive threat model is reviewed in [26], and a monitoring architecture is introduced. [2] provides a detailed security analysis of the issue of energy theft. The authors explained that AMI will significantly increase the risk of energy theft due to the interconnected nature of the infrastructure and the large-scale deployment of identical devices, leading to an amplification of effort, a division of labor, and an extended attack surface. The same authors introduce a penetration testing method to evaluate AMI components in [3], where they reveal vulnerabilities such as passwords sent in clear over optical ports, the possibility of replaying authentications, and the derivation of encryption keys from meter passwords.

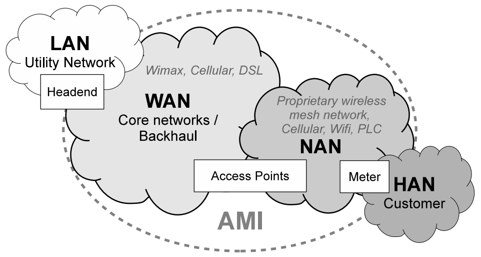

Fig. 1. Overview of the AMI Networks

An architecture called the cumulative attestation kernel that addresses the issue of securely auditing firmware updates in embedded systems such as smart meters is introduced in [27]. The system is designed to be cost-, power-, computation-, and memory-efficient. A prototype is implemented to demonstrate the feasibility of the solution as well as to formally prove that it meets remote attestation requirements. In the field of intrusion detection, [28] proposes a model-based sensor working on top of the WirelessHART protocol to monitor and protect wireless process control systems. The hybrid architecture consists of a central component that collects information periodically from distributed field sensors. A set of eight detection rules working on the physical, data-link, and routing layers covers threats including signal jamming, node compromise, and packet modification.

III. Background and Approach

The role of an AMI is to support two-way communication between meters and utilities. It does so through two main types of communication networks: wide-area networks (WAN) and sets of neighborhood-area networks (NAN). As shown in Figure 1, meters are deployed in the NAN and connect to their local access points to exchange information with the collection engine back in the utility network. The WAN often uses long-range and high-bandwidth communication technologies, such as Power Line Carrier (PLC), cellular technologies, or the emerging WiMAX. The NAN has shorter range and lower bandwidth requirements, and a proprietary radio frequency carrier is often used to connect meters in a mesh network, although some AMIs are deployed with PLC or with meters that have direct cellular communication capabilities. Access points act as gateways between these different communication technologies. Meters and access points hold a microcontroller unit and a flash memory unit, a datastore, and one or more communication modules. Meters also carry a metrology unit to measure power consumption. AMI components could also include repeaters, to support long-range communication, and tools used by field crews to manage the network. The collection engine in the headend is in charge of sending periodic data requests to meters, storing responses in a meter data management system, sending configuration and price information, and receiving alerts such as outage information.

To understand the threats faced by this infrastructure, we previously reviewed the possible attacker motivations and the different categories of vulnerabilities that could be exploited [26]. Vulnerabilities can arise from flaws in or misuses of the different network protocols (routing, configuration, name resolution, encryption, or authentication protocols), or from flaws in hardware or software designs and implementations (e.g., read and write access to data storage, or access to encryption keys). We refer the reader to [26] for a discussion on possible attack techniques, along with their direct consequences. Identifying attack consequences helps us reason about the various security solutions that need to be deployed to ensure a resilient and secure infrastructure.

Current security solutions to protect AMIs usually include physical controls (e.g., tamper-resistant seals on meters), meter authentication and encryption of all network communications, and network controls (firewalls are deployed at the access points and in front of the headend). On the security detection side, IDSes are usually deployed inside the utility network to identify attacks against the headend. This means that current intrusion detection solutions for AMI are from a central location, where they can suffer from scalability issues. (A large-scale AMI can reach several million nodes.) More importantly, security administrators have no ability to see the traffic among meters at the edge of the NAN, and they have to rely on encryption, secure key storage, and the use of protected radio frequency spectrum to prevent intrusions.

The objective of this work is to address those challenges and to bridge the gap between existing security solutions and the threat model reviewed in [26]. We focus specifically on designing a monitoring solution for the NAN that can identify compromised meters and malicious network activity in near real-time. This intrusion detection solution must be able to detect both known and unknown threats with high accuracy while having a minimal impact on the network infrastructure and network management operations. It is important to understand that unlike traditional IT organizations, which are most concerned with confidentiality and integrity of systems and information, the top priority for utilities is availability. Unavailabilities of human operators or control systems could lead to a possible disruption in the energy delivery mission. Consequently, security solutions should be seamlessly integrated into the existing infrastructure; in particular, IDSes should generate very few false positives.

We believe that a good approach to meeting those requirements would be to develop a specification-based intrusion detection sensor that can be deployed in the field to listen passively to meter traffic. Relying on specification rather than on attack signatures enables the sensor to model valid meter behavior so that unknown attacks can be detected, and, more importantly, so that formal methods can be leveraged to make sure that detection rules are sufficient to cover a given security policy [24]. Moreover, instrumentation of dedicated sensors in the NANs makes it easier to reach high scalability while not being limited by the low computational capabilities of meters or access points. We introduce the system model and security requirements in the next section, and we present the formal verification framework in Section V.

IV. System Model and Security Requirements

In this section, we present the development of a set of security requirements based on the protocol and device specifications. These security requirements will be used in the next section to build a formal model of the monitoring solution and the security policy. The policy is high-level and defines how we expect the global network to behave so that malicious activity can be accurately detected. The security requirements are designed at a lower level and provide constraints that will be monitored locally by intrusion detection sensors to guarantee that the security policy is respected everywhere. Security requirements are based on 1) the threats model from [26], 2) an analysis of the communication protocol specifications and expected behavior of meters, and 3) historical training data from expected use cases. Specifically, we follow the systematic framework introduced by [29] to develop a hierarchical set of specifications and constraints from the system model and the threat model. We start by presenting the different protocols used by AMI devices and then we define a system state machine that we translate into security specifications.

A. AMI Protocols

The design and deployment of open standards play an important role in the smart grid effort. Utilities and vendors have all agreed that interoperability is a key requirement for success in upgrading the infrastructure and deploying new components. As a result, several standard protocols are currently implemented in most AMI devices. In the US, the ANSI C12 series of standards seems to predominate and is the focus of this study, while in the EU, the IEC62056 family of protocols leads the AMI market. The two main C12 standard protocols that we analyzed are the following.

- *ANSI C12.22: communication over any reliable network [30].* This standard defines the application-layer protocol to transfer data tables over a reliable network such as TCP/IP. C12.22 datagrams are made of three parts: 1) an Association Service Control Element (ASCE) header that defines the security context, 2) a set of service requests or replies defined according to the Extended Protocol Specification for Electric Metering (EPSEM), and 3) an optional data payload structured according to the C12.19 protocol introduced below.
- *ANSI C12.19: data table definition [31].* This standard defines the data structure for use in transferring data to and from devices via the various C12 communication protocols. The tables are organized into decades based on feature-set, and the standard defines a total of 164 tables and 16 decades. For example, decade 11 holds tables to store load control and pricing information while decade 12 holds tables for network control information.

Two additional C12 communication protocols have been defined but are beyond the scope of this work: the ANSI C12.18, point-to-point communication over optical connections, and the ANSI C12.21, communication over telephone modems. The former is used by field crews to install or repair meters through their front-facing optical ports while the latter replaces C12.22 in some AMI deployment. For the remaining parts of this study, we assume that we are working with an AMI deployed according to Figure 1 and with a cellular-based WAN and a radio-frequency mesh network for the NAN, both using C12.22 over TCP/IP to transport C12.19 tables. We note that our specification definition method could be easily adapted to alternative configurations.

B. System Model

Devices within an AMI are expected to exhibit certain behavior. Meters are installed, configured, and then periodically queried by the collection engine to retrieve energy consumption or diagnostic information. To capture a legitimate activity profile for these interactions, we start by developing a hierarchy of state machines at the network, device and application levels. In the next subsection, we complement these state machines with valid network traces to build a comprehensive system model of legitimate behaviors.

The expected behavior of meters at the network level is identified with respect to the specifications of the dynamic configuration protocol and the routing protocol (layers 2 and 3 of the OSI model). Meters being deployed in a wireless mesh network must dynamically receive configuration and routing information. Specification-based intrusion detection solutions for this type of protocols, such as DRCP or AODV, have been extensively studied [24], [14], [16], so we focus our effort on upper layers.

Then, we consider a device-level state machine. Figure 2 represents the three possible states for meter devices: "off-line," "in use," and "to configure." While very simple, this state machine is important, because it will drive the set of constraints monitored by the IDS sensors. The three states define different functionality and permission levels. A meter "in use" has full operational capabilities, while a meter "to configure" is limited to receiving configuration information and should never interact with other devices for other operations. Finally, the network activity for a meter "offline" should be non-existent. This separation helps to isolate meters that do not respect all the security requirements. For example, a meter with an unknown identification number would fall directly into the "to configure" state, because its configuration does not match the requirement to have all meters in use be properly registered in the collection engine. Similarly for a meter that does not properly respond to network requests. As a result, spoofing attack or compromised meter can be identified and further investigated. To transition from the state "to configure" to the state "in use," a remote attestation system such as [27] can be deployed to ensure that meters are running a valid firmware and are properly configured.

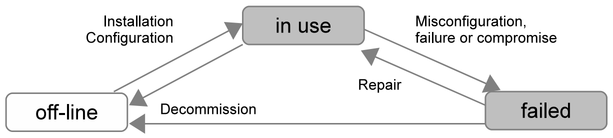

Fig. 2. Device-level state machine for smart meters. These high-level states guide the set of valid behaviors monitored by the IDS sensors at a lower level.

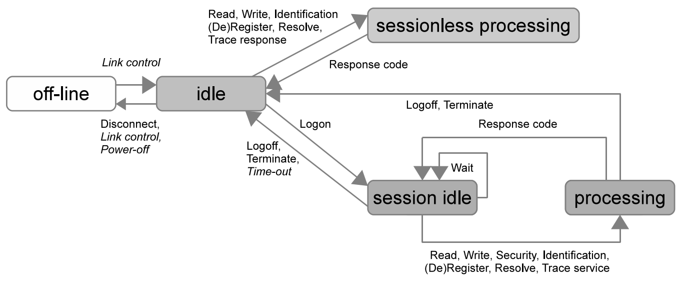

Fig. 3. Application-level state machine for the C12.22 standard protocol, including both client and server behaviors when the device is "in use." If the device is in "to configure" state, then we assume that it is limited to receiving configuration information.

Finally, we define a state machine at the application level for the C12.22 protocol. Figure 3 represents an extension of the state machine defined in the ANSI standard [30]. We note that communications can be done with or without a session. A session means that a valid authentication procedure is done, and we use it as a constraint for access and operational restrictions.

C. Constraints and Security Requirements

We developed a hierarchical set of constraints that will be monitored by the AMI intrusion detection sensors. The hierarchy was introduced by [29] and covers three constraint categories: network-based, device-based, and application-based constraints. For each of these categories, the framework covers five constraint types: data, access, timing, resource usage, and operational constraints. We define constraints for each of these categories and types based on analysis of the system model, the potential threats, and network traces captured for a variety of use cases. AMI use cases were extracted from [32] and include meter reading, meter event, service switch (i.e., remote connect and remote disconnect), price information, prepayment configuration, islanded distribution customer storage (i.e., to cap meter load), outage notification, direct load management, premise network administration (e.g., to request that a new home appliance become part of the HAN), and firmware and program updates. We then used a meter emulator called *Table TstBench* to generate valid C12.22 traces for most use cases.

In the network category, operational constraints are defined by the protocols at the network and transport layers: TCP/IP and configuration and routing protocols. They aim to detect low-level abuses such as communication interception through man-in-the-middle attacks. The access constraints are inferred from the network topology and access control rules. For example, meters in a given NAN can connect to a single access point and should never initiate connections to an access point in another NAN. We note that for a large network, the network access control lists can be extracted automatically using tools such as NetAPT [33]. Data, timing, and resource usage constraints are captured from network traces using a solution described in the application category.

In the device category, operational constraints are defined by the device-level state machine illustrated in Figure 2 and a set of configuration requirements defined by the utility. We assume that a template for valid meter configuration is created when an AMI is deployed. One can then check the configuration remotely by reading the C12.19 tables and by attesting the firmware. The aim of the device-level operational constraints is to detect compromised meters, configuration abuses, or incorrect meter upgrade operations. We do not cover data, access, timing, and resource usage constraints at the device level, because the internal operating system behavior of the device is not visible to the intrusion detection sensor. We note that these constraint types could be included in the monitoring solution if meters and access points could periodically send system-health reports.

In the application category, operational constraints are defined by the application-level state machine (illustrated in Figure 3), which is dependent on the devices being in a valid "in use" state. Constraints for data, access, timing, and resource usage in the application category, and constraints for data, timing, and resource usage in the network category, are inferred from a combination of use requirements and network traces, using the following two-step approach.

First, for each use case, a set of traces is recorded and parsed to extract the following features: source and destination end points, distribution of the number of bytes per session, distribution of session durations, sequence of C12.22 services requested, set of C12.19 tables read and written, and, finally, the data values read from and written to those C12.19 tables.

Second, the results from that initial parsing phase are reviewed and adjusted by an operator to define actual constraints. For example, if the distribution of the number of bytes for a given use case goes from 76 to 98 with a mean value of 87 bytes, then the operator can establish the rule that any session that belongs to this use case and would have a number of bytes larger than 100 would be illegitimate. The goal of the approach is to build generic constraints from specific traces while minimizing human involvement and effort. Features are predefined from the network and application protocols, so the process of integrating new AMI use cases is simple.

To illustrate the approach, Table I presents requirements extracted from [32] for a few use cases. Each of these use cases is made of several payloads. We captured network traces for some of these payloads using the testbed described in Section VII, and we analyzed these traces using our constraint extraction script. Table II shows the results of this procedure on a single payload, listing features and values obtained after the automated and manual steps for data, access, timing, and resource usage constraints in the network and application categories. The aim of the constraints is to detect meter compromises, traffic modification, injection and replay, spoofing of network nodes, authentication violations, and denial-of-service attacks.

V. Formal Verification

In this section, we develop a security policy and apply the formal framework from [24] to verify that critical monitoring operations affecting the global security policy are correctly designed. It is important to understand that among the different constraints introduced, some require IDS sensors to exchange information in order to monitor behaviors at the global scale. For example, checking that the frequency of occurrence of a given use case does not go beyond a required limit is a distributed operation. We decided to implement a formal model that we can mathematically prove to validate this type of operation. We start by introducing the framework and the theorem prover before describing the policy and the different formalisms required to conduct the proof.

A. Overview of the Formal Verification Framework

The key idea of the formal framework is to build a model of the security-relevant attributes of the system and then formally verify that no network trace can violate the security policy without being detected. The framework introduced by [24] is built hierarchically in five layers: 1) a model of the network, 2) a set of monitoring operations and a set of assumptions (to cover unmonitored items), 3) the protocol specifications, 4) the security policy, and 5) a verification theorem. The theorem verifies whether all possible network traces that respect the first three layers of the framework will also respect the fourth layer.

The five layers are implemented as functions and data structures in ACL2, which is a software tool combining a programming language, a logic, and a theorem prover based on Common Lisp [34]. ACL2 automates most of the proof effort using techniques such as rewriting and mathematical induction, but one often needs to guide the proof by adding lemmas that the mechanical prover cannot deduct by itself. We note that this manual operation is very useful for discovering hidden assumptions and incomplete functions. We provide a description of the formalism in the remaining parts of this section.

B. Network Model

The network model consists of a set of nodes and an ordered sequence of network traces. Nodes are defined by a data structure that includes attributes such as IP address, current application state, and log information (e.g., counters for the numbers of requests received and responses sent per use case). Network traces are represented by a flow structure that includes attributes such as source and destination IP addresses, duration, number of packets, number of bytes, and sequence of C12.22 service requests and responses. We note that time is modeled in relation to each flow through the duration attribute and not as an absolute value for the network. This greatly simplifies the model by removing the need to run a global

TABLE I

EXAMPLE OF USE CASE REQUIREMENTS FOR COMMUNICATIONS BETWEEN COLLECTION ENGINE AND METERS, EXTRACTED FROM [32]

Use case	Payload	Frequency	Resp. time
Meter reading	reading request for multiple intervals	25 per 1000 meters per day	$<$ 15 sec
Service switch	scheduled cancel service switch request	1-2 per 1000 meters per day	$<$ 1 min
Outage notification	outage notification message	100% of meters losing power	$<$ 20 sec
Firmware upgrade	metrology firmware update request	1-2 per meter per year	$<$ 4 hours

TABLE II

NETWORK-LEVEL AND APPLICATION-LEVEL CONSTRAINTS INFERRED FROM A SERVICE SWITCH PAYLOAD TRACE (I.E., REMOTE DISCONNECT PROCEDURE)

Type	Feature	Automated Results	Manual Correction
		Network-level	
Access	nodes	from Headend to Meter	*Unchanged*
Data	protocols	C12.22 over TCP/IP	*Unchanged*
Timing	frequency	-	**1-2 per 1000 meters per day**
Resource usage	Session size	87 bytes	**100 bytes**
		Application-level	
Access	C12.19 tables	Table 0 (read), Table 3 (write)	*Unchanged*
Data	C12.19 values	Table 3, Data: "01", Offset: "0x00"	Data: "**binary value**"
Timing	Session duration	0.78 second	**$<$ 1 min**
Resource usage	Services used	Logon (0x50), Full Read (0x30), Partial Write (0x4f), Logoff (0x52)	*Unchanged*

clock. The model runs the following recursive function to process flows and to update nodes:

```
(defun process_flows (flowlist nodelist sys)
  (if (endp flowlist)
    (valid_node_use flowlist nodelist sys)
    (and
     (process_flow  (car flowlist) nodelist sys)
     (process_flows (cdr flowlist)
        (update_node nodelist (car flowlist)) sys)
     )))
```

The function *process_flow* handles a single flow and is responsible for applying the monitoring operations that we describe next.

C. Application Protocol and Local Monitoring Operations

The next two layers of the formal verification framework are a model of the application-level protocol and an implementation of local monitoring operations. The function *process_flow* checks the validity of an individual flow following the constraints introduced in the previous section:

```
(defun process_flow (flow nodelist sys)
  (if
    (and
     (flow-p flow)
     (equal (run_state_machine flow) 'idle)
     (equal (flow-sip flow) (sys-mdms_ip sys))
     (valid_frequency_use flow nodelist sys)
    (>= (sys-limit_response_time sys)
         (flow-response_time flow)))
    t
    nil
    ))
```

We integrated four monitoring rules into the verification. The rules check the validity of the flow format, the validity of the sequence of C12.22 services through an application-level state machine, the validity of the flow origin, the validity of the use case frequency, and the validity of the flow duration. The function *valid_state_machine* is a recursive function that checks application state transition for a given flow. For example, one constraint is that the *write* service is accessible only after a valid *security* service has been performed. This security service returns a valid acknowledgment only if the source of the flow has the correct authentication credentials. We note that the current model does not include an implementation of the authentication process, so the credential-checking operation is assumed to be correct.

Other assumptions include 1) complete visibility of the meter traffic of a given NAN to the intrusion detection sensor deployed in the NAN, 2) reliability of the collection engine, 3) reliability of the configuration and routing protocols, and 4) perfect communication among intrusion detection sensors in recording and sharing high-level meter activity (e.g., use case frequency is recorded per meter, but the information is shared among all sensors). The second assumption does not mean that the collection engine cannot be spoofed, but it indicates that there will always be a valid collection engine in the network. The third assumption can be removed if we include the processing at the network and transport layers in the model, but the focus of the current verification effort is on the application layer, and lower layers are left as future work.

D. Global Security Policy

The last two layers of the formal verification framework are the security policy and the verification theorem. The security policy includes rules to check that no network trace in the entire AMI contains illegitimate activity. The current proof effort covers the following four key rules:

1) *Only well-formed C12.22 requests and responses are authorized at the application layer*
2) *Only the collection engine can initiate C12.22 requests to meters*
3) *Requests labeled as sensitive (e.g., remote disconnect) cannot target a rate of more than Y meters per hour*
4) *Correctly configured and working meters respond to requests in less than Z seconds*

The first rule is designed to detect traffic modification, injection, and replay. The second rule helps to identify compromised meters. The third rule prevents malicious nodes from disrupting a large set of meters. Finally, the last rule supports the detection of denial-of-service attacks. The objective of the proof is to check that any network trace that passes the local monitoring operations would comply with the global security policy. In other words, there is no network trace that can violate the security policy while going undetected by the intrusion detection sensors. The proof is implemented through the following theorem:

```
(defthm local_to_global_verification
  (implies
   (and
    (nodelistp nodelist)
    (sys-p sys)
    (process_flows flowlist nodelist sys)
   )
   (check_policy flowlist)))
```

The function *check_policy* includes four recursive functions to check the four rules in the policy. The important concept to understand is that the function *process_flows* represents the local intrusion detection sensors, while the function *check_policy* represents a global view of the AMI network. It would be impractical to capture that global view, because that would require sharing of network traces among all the intrusion detection sensors. The goal of the theorem is to prove that distributing the monitoring operations among sensors still guarantees that the global security policy is enforced.

For example, the first rule in the current policy states that only well-formed C12.22 flows are authorized in the network. The following two functions implement this rule:

```
(defun valid_protocol_check (flow)
  (if (equal (run_state_machine flow) 'idle)
    t
    nil)
  )

(defun valid_protocol (flowlist)
  (if (endp flowlist)
    t
    (and
     (valid_protocol_check (car flowlist))
     (valid_protocol (cdr flowlist))
  )))
```

The function *valid_protocol* is recursive and processes a list of flows, while the function *valid_protocol_check* processes individual flows and checks the correct sequence of C12.22 services through the application-level state machine. The fact that the same function *run_state_machine* is used to validate the global security policy and the local monitoring operations makes the proof of this rule trivial. On the other hand, the third policy rule, which checks the correct frequency of use cases, is much more complex to prove, because the local monitoring operations rely on counters updated for individual meters, while the global security policy looks at network traces as a whole. The following ACL2 traces summarize the proof for the third rule:

```
Summary
Form:   ( DEFTHM LOCAL_TO_GLOBAL_RULE-3  ...)
Rules:  ((:DEFINITION CHECK_RULE_3)
         (:DEFINITION ENDP)
         (:DEFINITION NOT)
         (:DEFINITION PROCESS_FLOWS)
         (:DEFINITION SYNP)
         (:DEFINITION UPDATE-NETWORK)
         (:EXECUTABLE-COUNTERPART BINARY-+)
         (:EXECUTABLE-COUNTERPART UPDATE-NETWORK)
         (:FAKE-RUNE-FOR-LINEAR NIL)
         (:FAKE-RUNE-FOR-TYPE-SET NIL)
         (:INDUCTION CHECK_RULE_3)
         (:INDUCTION PROCESS_FLOWS)
         (:REWRITE |(+ 0 x)|)
         (:REWRITE |(< (if a b c) x)|)
         (:REWRITE LEMMA1)
         (:TYPE-PRESCRIPTION CHECK_RULE_3))
```

This proof often relies on multiple lemmas to succeed. These lemmas were introduced manually to guide the mechanical prover of ACL2. Each lemma is a property of the model. Some of these properties are relevant to the security of the system. For instance, *lemma1* proves that the function *process_flows* correctly update meters information to keep track of the frequency of use cases:

```
(defthm lemma1
  (implies
   (and
    (nodelistp nodelist)
    (sys-p sys)
    (integerp (sys-limit_sensitive sys))
   (< 0 (sys-limit_sensitive sys))
    (consp flowlist)
    (process_flows flowlist nodelist sys))
  (> (sys-limit_sensitive sys) (count_frequency_sensitive
                                  nodelist))
   ))
```

We note that in our experience, implicit assumptions made by model developers are the main reason proof attempts fail. Going through the proof exercise is critical to understanding where the model is incorrect or what is missing from the monitoring operations that enforce the security policy.

VI. Implementation

For evaluation purposes, we implemented a prototype of the intrusion detection sensor in Python on a Linux platform. Figure 4 shows the different components of the prototype. A dissector receives network packets captured through the *pylibpcap* library[1] and extracts values from the different fields at the network layer (IP header), transport layer (TCP header), and application layer (C12.22 payload). A parser then interprets these values according to the protocol specifications and reports to a format validation module and a state machine module. The former monitors stateless constraints, while the latter keeps track of the state of each device for which traffic was captured, to make sure that stateful constraints are not violated. In our current prototype, constraint violations are reported to a central intrusion detection server that stores alerts and presents results to security analysts. This configuration may not be optimal in a large-scale AMI deployment, and a decentralized intrusion detection structure could increase the reliability of the IDS. We leave the investigation of such questions to future work.

[1]http://pylibpcap.sourceforge.net/

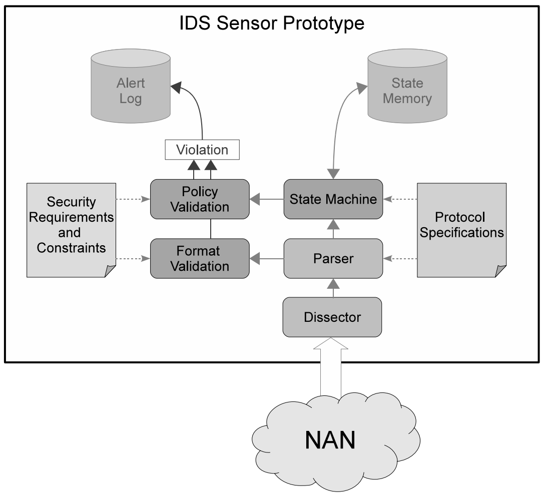

Fig. 4. Diagram of the internal modules of our IDS sensor prototype

VII. Evaluation and Discussion

Two goals drive the empirical evaluation of the intrusion detection sensor: 1) verifying that the implementation is correct, and 2) measuring the performance of the implementation under various conditions. The first objective is critical because it complements the formal approach. Formally proving that the specifications are complete with respect to the defined security policy only validates the part of the system that we modeled, and it does not prevent implementation defects from introducing vulnerabilities in the sensors. Running the prototype implementation in a testbed contributes to reduce the likelihood that critical vulnerabilities be left unnoticed.

A. AMI Testbed

To create a realistic AMI environment where we can easily run and reproduce experiments, we built meter emulators in virtual machines. We used an industry software called Table TstBench developed by Trilliant[2] to emulate C12.22 and C12.19 standard protocols on top of a TCP/IP communication module. A first virtual machine holds multiple emulators in "server mode" (i.e., to behave as meters) and a second virtual machine holds an emulator in "client mode" (i.e., to behave as a collection engine). To reproduce the context of a NAN with a mesh network, these emulators are all connected to the same subnet and free to communicate with any device. A third virtual machine was deployed to capture all network traffic and to host the intrusion detection sensor.

The machine hosting the testbed has an Intel Xeon with 8 cores at 2.67 GHz and 12 GB of RAM. The operating system is Ubuntu 10.04 and VirtualBox is used as virtualization solution. The virtual machines are all configured with 1 GB of RAM.

B. Test Harness

To evaluate the efficiency and correct implementation of our prototype, we used the testbed to generate two datasets: a set of valid use cases and a set of attacks. The two datasets are described in Table III. The *meter-reading* attack consists of issuing multiple read requests towards all C12.19 tables. This attack violates the following constraints: resource usage at the network layer (i.e., number of bytes), access and data at the application layer (i.e., set of C12.19 tables and data values read), and resource usage at the application layer (i.e., sequence of C12.22 services requested). The *service swith* attack consists of enabling and then disabling service in rapid succession at a given meter. It violates the following constraints: resource usage and timing at the network layer (i.e., number of bytes and frequency), and resource usage at the application layer (i.e., sequence of C12.22 services requested). Each C12.22 transaction from those two datasets was recorded as a PCAP trace, and we developed a script to replay the different traces in various combinations and at multiple frequencies using Tcpreplay[3]. The output of the sensor was monitored after each test, and we recorded the following metrics: CPU usage, memory usage (resident set size), numbers of packets and bytes processed per second, and correct identification of valid use cases and malicious attacks.

C. Evaluation Results

1) Correctness:: The 3 uses cases and 2 attacks from Table III were tested individually and in combination to validate the correctness of the constraints and security requirements that were implemented. The meter reading routine was replayed sequentially with various destination IP addresses to replicate the behavior of a collection engine sending energy consumption requests to an entire subnet. Legitimate service switch and load management requests as well as attacks were injected in parallel of this routine. The two attacks were also replayed with multiple destination IP addresses to simulate a compromised meter sending malicious payload to several targets. The source IP address of attacks was also modified to simulate a malicious device spoofing the meter data management system. Out of 644,757 packets and 32,367 connections processed, the results report 100% correct identification of malicious connections, and 99.57% of correct identification of legitimate use cases. The few false positives helped to reveal a weakness of the sensor when dealing with concurrent C12.22 operations exchanged between two devices through the same TCP source and destination ports. This issue comes from the method used by the sensor to identify flows and we are working to fix it.

2) Performance:: We measured that under the average conditions of around 10 packets processed per second with a network of 20 meters, the sensor uses 0.3% of the CPU of our virtual machine and 10 MB of memory. We then proceeded to test the CPU and memory usage under more extreme conditions. The first test consisted of measuring the CPU usages for various rate of packets sent between two devices. Figure 5 shows the CPU usage results for both the packet dissector module and the state machine module of the sensor for up to 240 packets processed per second, which

[2]http://www.trilliantinc.com/

[3]http://tcpreplay.synfin.net

TABLE III
DATASETS GENERATED USING TABLE TSTBENCH AND RECORDED AS PCAP TRACES FOR TESTING

Name	Description	packets
Valid Use Cases		
Meter reading	Reading request for consumption registers	16
Service switch	Remote disconnect request	19
Load management	Update load management profile request	33
Malicious Behavior		
Meter reading	Multiple reading of all C12.19 tables	49
Service switch	Multiple connect and disconnect requests	36

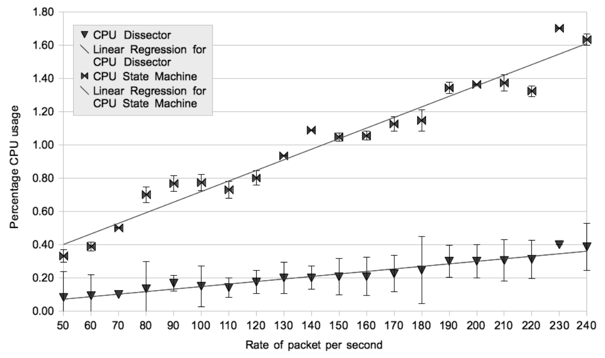

Fig. 5. Evolution of CPU usage for the packet dissector and the sensor state machine with respect to the rate of packet per second

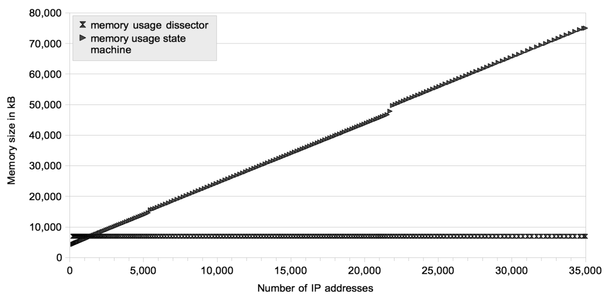

Fig. 6. Evolution of memory usage for the packet dissector and the sensor state machine with respect to the of IP addresses monitored

represented 13 MB of network traffic per second. The CPU usage for both modules increases linearly, but the CPU usage for the state machine increases faster due to the larger number of computations performed on each packet. The second test consisted of measuring the memory usage for the meter reading operation sent to and from an increasing number of devices. Figure 6 shows results for up to 35,000 IP addresses monitored. We note that to be conservative, data structures holding the state machine for each node were not expired and were kept in memory for the entire duration of the experiment. The results reveal that the memory usage of the dissector is not affected by the size of the network monitored, while the memory usage of the state machine grows linearly and reaches a maximum of 76 MB.

D. Discussion

These results indicate that the performances of the implementation are compatible with the deployment of the sensor on embedded devices, and help to understand the scalability of the solution to monitor large network or to sustain significant bandwith requirements. It is important to note that on the one hand, the code has not yet been optimized and is not implemented in a low-level programming language. On the other hand, the sensor does not yet handle packet encryption, which is the norm in AMI deployment and will not only add overhead to the packet dissector module but will also require decryption keys to be shared between meters and intrusion detection sensors. We leave the investigation of this issue and how it affects performance for future work.

VIII. CONCLUSION

We introduced in this paper a specification-based intrusion detection sensor dedicated to monitoring AMI communication traffic. The solution relies on protocol specifications, security requirements, and a comprehensive security policy to detect security violations and trigger alerts when suspicious activity occurs. The security requirements were developed using a systematic approach that combined a threat model, a system model, and a set of network traces that capture valid use case behavior. We then leveraged a formal verification framework to guarantee that any event that could compromise the security policy at the application layer would be identified. Among other things, it prevents a compromised meter from sending remote disconnect commands to a large number of targets. Finally, we deployed a prototype implementation of the sensor in a realistic AMI environment and tested the efficiency of the solution under various conditions.

The next steps on this project will be to develop an intrusion detection infrastructure to integrate the sensors within a complete situational awareness solution, and to test the implementation further with hardware meters in a live AMI installation. We are also interested in studying how automated response and recovery actions could be added to the architecture.

ACKNOWLEDGEMENTS

We thank Trilliant Inc. for their support with Table TstBench. This material is based upon work supported by the Department of Energy under Award Number DE-OE0000097.

References

[1] N. Falliere, L. Murchu, and E. Chien, "W32. Stuxnet Dossier," *Symantec Security Response*, 2010.

[2] S. McLaughlin, D. Podkuiko, and P. McDaniel, "Energy theft in the advanced metering infrastructure," *Critical Information Infrastructures Security*, pp. 176–187, 2010.

[3] S. McLaughlin, D. Podkuiko, S. Miadzvezhanka, A. Delozier, and P. McDaniel, "Multi-vendor penetration testing in the advanced metering infrastructure," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 107–116.

[4] T. Song, C. Ko, J. Alves-Foss, C. Zhang, and K. Levitt, "Formal reasoning about intrusion detection systems," in *Recent Advances in Intrusion Detection*. Springer, 2004, pp. 278–295.

[5] C. Ko, M. Ruschitzka, and K. Levitt, "Execution monitoring of security-critical programs in distributed systems: A specification-based approach," *sp*, p. 0175, 1997.

[6] R. Sekar and P. Uppuluri, "Synthesizing fast intrusion prevention/detection systems from high-level specifications," in *USENIX Security Symposium*. Citeseer, 1999, pp. 63–78.

[7] P. Uppuluri and R. Sekar, "Experiences with specification-based intrusion detection," in *Recent Advances in Intrusion Detection*. Springer, 2001, pp. 172–189.

[8] M. Raihan and M. Zulkernine, "Detecting intrusions specified in a software specification language," in *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, vol. 1. IEEE, 2005, pp. 143–148.

[9] C. Ko, "Logic induction of valid behavior specifications for intrusion detection," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2002, pp. 142–153.

[10] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, Y. Wang, X. Wang, and N. Stakhanova, "Software fault tree and coloured Petri net–based specification, design and implementation of agent-based intrusion detection systems," *International Journal of Information and Computer Security*, vol. 1, no. 1, pp. 109–142, 2007.

[11] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt, "Using specification-based intrusion detection for automated response," in *Recent Advances in Intrusion Detection*. Springer, 2003, pp. 136–154.

[12] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-based anomaly detection: a new approach for detecting network intrusions," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 265–274.

[13] N. Stakhanova, S. Basu, and J. Wong, "On the symbiosis of specification-based and anomaly-based detection," *computers & security*, vol. 29, no. 2, pp. 253–268, 2010.

[14] C. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. Levitt, "A specification-based intrusion detection system for AODV," in *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*. ACM, 2003, pp. 125–134.

[15] E. Hansson, J. Groenkvist, K. Persson, and D. Nordqvist, "Specification-Based Intrusion Detection Combined with Cryptography Methods for Mobile Ad Hoc Networks(Policy-Baserad Intrangsdetektering foer Mobila Ad Hoc-Naet)," NASA Center for AeroSpace Information, 7121 Standard Dr, Hanover, Maryland, 21076-1320, USA, Tech. Rep., 2005.

[16] H. Hassan, M. Mahmoud, and S. El-Kassas, "Securing the AODV protocol using specification-based intrusion detection," in *Proceedings of the 2nd ACM international workshop on Quality of service & security for wireless and mobile networks*. ACM, 2006, pp. 33–36.

[17] C. Tseng, T. Song, P. Balasubramanyam, C. Ko, and K. Levitt, "A specification-based intrusion detection model for olsr," in *Recent Advances in Intrusion Detection*. Springer, 2006, pp. 330–350.

[18] P. Truong, D. Nieh, and M. Moh, "Specification-based intrusion detection for H. 323-based voice over IP," in *Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on*. IEEE, 2005, pp. 387–392.

[19] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia, "VoIP intrusion detection through interacting protocol state machines," 2006.

[20] T. Phit and K. Abe, "A Protocol Specification-Based Intrusion Detection System for VoIP and Its Evaluation," *IEICE Transactions on Communications*, vol. 91, no. 12, pp. 3956–3965, 2008.

[21] P. Jieke, J. Redol, and M. Correia, "Specification-Based Intrusion Detection System for Carrier Ethernet," in *Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST)*. Citeseer, 2007.

[22] R. Gill, J. Smith, and A. Clark, "Specification-based intrusion detection in WLANs," 2006.

[23] T. Song, J. Alves-Foss, C. Ko, C. Zhang, and K. Levitt, "Using acl2 to verify security properties of specification-based intrusion detection systems," in *Fourth International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2-2003)*. Citeseer, 2003.

[24] T. Song, C. Ko, C. Tseng, P. Balasubramanyam, A. Chaudhary, and K. Levitt, "Formal reasoning about a specification-based intrusion detection for dynamic auto-configuration protocols in ad hoc networks," *Formal Aspects in Security and Trust*, pp. 16–33, 2006.

[25] F. Cleveland, "Cyber security issues for advanced metering infrastructure (AMI)," in *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*. IEEE, 2008, pp. 1–5.

[26] R. Berthier, W. Sanders, and H. Khurana, "Intrusion Detection for Advanced Metering Infrastructures: Requirements and Architectural Directions," in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE, 2010, pp. 350–355.

[27] M. LeMay and C. Gunter, "Cumulative attestation kernels for embedded systems," *Computer Security–ESORICS 2009*, pp. 655–670, 2010.

[28] T. Roosta, D. Nilsson, U. Lindqvist, and A. Valdes, "An intrusion detection system for wireless process control systems," in *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*. IEEE, 2008, pp. 866–872.

[29] C. Ko, P. Brutch, J. Rowe, G. Tsafnat, and K. Levitt, "System health and intrusion monitoring using a hierarchy of constraints," in *Recent Advances in Intrusion Detection*. Springer, 2001, pp. 190–203.

[30] *ANSI C12.22: Protocol specification for interfacing to data communication networks*. National Electrical Manufacturers Association, 2008.

[31] *ANSI C12.19: Utility industry end device data tables*. National Electrical Manufacturers Association, 2005.

[32] O. S. G. user group (OpenSGug) Smart Grid Network Task Force, "Smart grid network system requirements specifications v5.0-draft1," http://osgug.ucaiug.org/UtiliComm, 2011.

[33] D. Nicol, W. Sanders, S. Singh, and M. Seri, "Usable global network access policy for process control systems," *Security & Privacy, IEEE*, vol. 6, no. 6, pp. 30–36, 2008.

[34] M. Kaufmann, P. Manolios, and J. Moore, *Computer-aided reasoning: an approach*. Springer Netherlands, 2000.