

INTRUSION DETECTION AND PREVENTION IN ADVANCED
METERING NETWORKS

BY

HUAIYU ZHU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Professor David M. Nicol

Abstract

Advanced metering infrastructure (AMI) is envisioned to be able to revolutionize the power grid and turn it into the “smart grid”. AMI, through the use of “smart meters” and high-speed networks, promises to strengthen both the stability and sustainability of the grid. The vision of AMI is to enhance and improve the grid by providing fine-grained control over pricing and usage to both the utility and the customers. The promise is so convincing that there have been rapid, large-scale deployments all over the world in a very short time. In this frenzy of excitement, security of AMI, an issue of utmost importance, may have been overlooked. In this work, we present our in-depth study of the vulnerabilities in AMI to cyber-attacks. We also propose a scalable, content-aware methodology to stop propagating malware which exploits the vulnerabilities of AMI to disrupt the operation of service. Towards this end we design and implement a host-based policy engine that examines both ingress and egress traffic to the AMI application layer. Policy engine rules may refer to the structure and behavior of the AMI protocol, and may also perform multi-stage analysis of data payloads and look for evidence that malicious content is carried, rather than data. Our experimental results show that the policy engine is promising in controlling the malicious traffic and introducing negligible performance overhead.

Acknowledgments

I would like to express my sincere gratitude to my adviser Professor David Nicol. He kept providing me with helpful suggestions and constant encouragement since the beginning of my graduate study.

I also wish to thank my parents and my wife for their endless support that helped me finish my graduate study and research work.

Contents

1. Introduction	1
2. Background and Related Work	4
2.1 ANSI C12.xx Standard.....	4
2.2 DLMS/COSEM Standard	4
2.3 AMI Security.....	5
3. AMI Vulnerability Assessment	7
3.1 Threat of DDoS.....	7
3.2 C12.22 Vulnerabilities	7
3.2.1 Trace Service Vulnerability.....	7
3.2.2 Resolve Service Vulnerability	9
3.2.3 Urgent Traffic Vulnerability.....	10
3.3 Proposed Solutions	11
4. N-Gram Analysis for Malware	14
4.1 Traffic Type Classifier	14
4.2 Evaluation	14
5. Policy Engine Design	17
5.1 System Architecture.....	17
5.2 Data Collection.....	18
5.3 Policy Rule Monitoring System	18
5.4 Logging System	19
6. Standard Protocol Analysis	20
6.1 Overview	20
6.2 Policy Rules	20
6.2.1 Syntactic Policy Rules.....	20
6.2.2 Semantic Policy Rules.....	20
6.2.3 Access Policy Rules.....	21
6.2.4 Communication Policy Rules.....	21
7. Entropy Analysis.....	22
7.1 Entropy of Network Traffic	22

7.2 Evaluation	22
8. Pattern Analysis	26
8.1 Patterns in ARM Code	26
8.2 Evaluation	26
9. Signature-Based Analysis	29
9.1 Overview of Signature-Based Analysis.....	29
9.2 Experimental Setup.....	29
9.3 Signature Selection and Policy Design	30
9.4 Error Analysis for Single Signature	32
9.5 Pattern Recognition Using Multiple Signatures.....	33
9.5.1 Independent Signatures Modeling and Analysis.....	33
9.5.2 Correlated Signatures Modeling and Analysis	34
9.5.3 Packet Size Sensitivity	39
9.6 Malware File Detection Using Signatures	40
10. Implementation and Performance	42
10.1 Implementation	42
10.2 Performance Evaluation.....	44
11. Conclusion.....	46
References	47

1. Introduction

“Smart meters” in the advanced metering infrastructure (AMI) report real-time usage statistics over data networks to the utility. These meters receive pricing information from the utility for a given time of use. The AMI provides the utility with better control and more efficient response times to detect problems. The smart meter provides the user with the ability to make informed decisions about usage. The idea is fast gaining popularity all around the globe. Building a smart grid was mandated with the Energy Independence and Security Act of 2007, resulting in a substantial investment worth 4.5 billion USD towards that purpose [1]. The Obama administration has called for 40 million smart meters to be installed in the United States over the next three years [2]. The drive is not limited to the United States alone, Europe follows closely with a law mandating smart metering by the year 2022 [3], and over the next few years, an additional 100 million smart meters will be added to the existing base of 40 million units worldwide.

The AMI is composed of a combination of several communication networks and devices. A high-level overview of AMI is provided in Figure 1. One of the major components is a management component called AMI headend that resides and operates in the utility’s central office. Communication between field devices and the headend are carried out over a WAN. At the edges of the WAN, there lie data collectors or concentrators that provide access and aggregation of metering data. Finally, there is a mesh network of smart metering devices forming an LAN (local area network) or NAN (neighborhood area network).

Given the magnitude and scale of AMI and the sizeable variety of manufacturers entering the business, standardization is essential to ensure interoperability and seamless integration with existing infrastructure. To this end, the American National Standards Institute (ANSI) has been focusing on defining standards for AMI. The most recent is ANSI C12.22 [4] which is an application-layer specification to allow transport of meter data over any networked connection including IP and cellular technologies. An alternative protocol suite IEC 62056 used by many Asian and European countries is a set of standards for electricity metering data exchange proposed by the International Electrotechnical Commission. They are the International Standard versions of the DLMS/COSEM [5] specification.

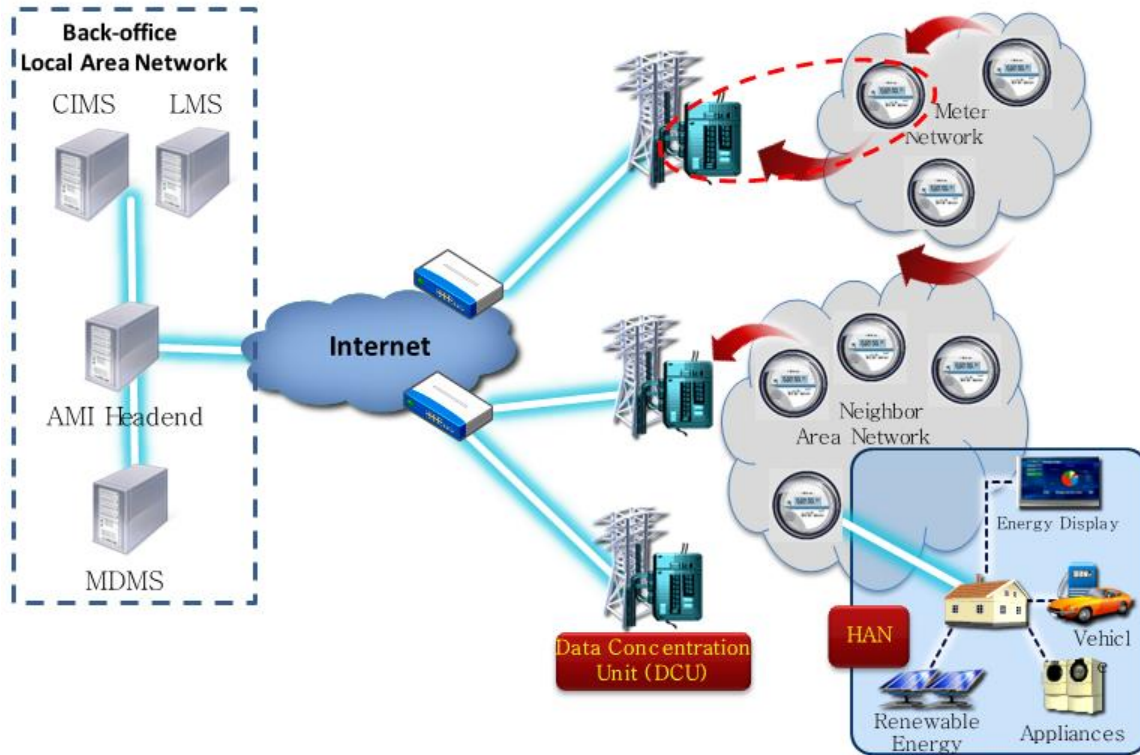


Figure 1 Architecture of the advanced metering infrastructure.

The complexity and magnitude of AMI is indicative of the presence of vulnerabilities. Vulnerabilities are weaknesses in the system that may expose crucial assets of the network to unauthorized access and unintended use. New levels of automation, combined with extended access to the grid, can provide unforeseen attack avenues to malicious users, and this poses a serious threat to the AMI because of the potential devastation that may result. Imagine a scenario where one or more hackers take control of some elements of the AMI and disrupt the power supply to residential areas or even worse, to sensitive facilities like hospitals or those containing nuclear assets.

The above-mentioned concerns have motivated researchers to solve many cyber-security issues, such as privacy, physical attacks, and data integrity problems. In this work, we focus on a particular threat from malicious software, called malware. The malware is able to destroy service availability and data integrity. Malware propagates through a network by hiding in the incoming/outgoing traffic in a system, sometimes as attachments to email, sometimes through direct attack on open services that have vulnerabilities. The main defense method against malware is to inspect whether the incoming or outgoing traffic includes malicious code, and to do that inspection at a low level in the protocol stack is required. The Bro system [6] for example can do deep packet inspection of traffic just as soon as it

enters a host, but is architecturally separated from application-level information that could be used as part of the inspection. This work proposes a host-based policy engine that has access to the network application states and so can do a more thorough analysis. In addition, the engine has the capability to do statistical analysis of data payloads to detect when the payload has characteristics of executable binaries.

The contributions of this thesis include:

- Conducted vulnerability assessment on the C12.22 protocol for the AMI system
- Created a DDoS (distributed denial of service) attack scenario based on C12.22 vulnerability and evaluated it with a simulation/emulation testbed
- Extended existing file type identification work to AMI traffic identification
- Designed a policy engine to inspect the AMI traffic
- Implemented a prototype of the policy engine based on an open source implementation of DLMS/COSEM provided by GuruX

2. Background and Related Work

2.1 ANSI C12.xx Standard

In this section, we provide a brief overview of ANSI C12.22 [4], a communication specification for transport of metering data.

Until recently, communication with electronic meters and other devices was done via manufacturer-dependent proprietary protocols. However, as the use of these devices became more and more widespread, ensuring interoperability became a necessity for seamless integration of devices and technologies. Interoperability requires standardization, and in the context of standardization for advanced metering, several standards including C12.18, C12.19, and C12.21 were proposed. The first step towards the standardization was the formulation of ANSI C12.19 (Utility Industry End Device Data Tables), which standardized data formats by abstracting data as a set of tables. This was followed by ANSI C12.18 (Protocol Specification for ANSI Type 2 Optical Port), that put forth a point-to-point protocol to transport table data over an optical connection through the use of an application language called Protocol Specification for Electric Metering (PSEM). PSEM allowed applications to read and write data tables. Following ANSI C12.18 and ANSI C12.19, ANSI C12.21 (Protocol Specification for Telephone Modem Communication) was developed, which allows devices to transport table data over telephone modems by using PSEM.

The newest among this chain of standards is the ANSI C12.22 (Protocol Specification for Interfacing to Data Communication Networks) [4] standard, which extends the usefulness and capabilities of C12.18, C12.19 and C12.21 to allow the transport of metering data (in protocol compliant format) over any reliable networking and communication technology.

The ANSI C12.22 standard specification provides a set of application-layer messaging services that are applicable for enterprises and end-devices of an AMI. The messages may be carried over a variety of existing underlying transport technologies including TCP-IP and UDP and also over a wide variety of physical media including PLC (power line communication), RF (radio frequency), etc.

2.2 DLMS/COSEM Standard

Similar to C12.22, DLMS/COSEM [5] is an application-layer protocol for communications and data exchange between the meters, the DCUs (data concentration unit), and the headend. The DLMS (Device

Language Message Specification) is a message specification, and the COSEM (Companion Specification for Energy Metering) is an interface model for communicating with the meters. The COSEM models the interface of the meters and the metering data with an Object Identification System (OBIS) defined in IEC 62056-61. The OBIS provides a unique identifier for each of the COSEM objects.

In the DLMS/COSEM protocol, each interface class has a global identifier, called its Class ID, and the object is instantiated in a uniquely recognizable OBIS code, called a logical name. The object has a set of attributes for the data and a set of methods for operations to be performed on the attributes. Each smart meter that is installed is modeled after a set of logical devices that act as a server application process (AP), and the logical device has a subset of the COSEM objects.

The DCUs are modeled as a set of application processes acting as a client application process (AP). The data exchange between a server AP and a client AP is performed through an application level connection, called Application Association (AA), which is always initiated by the client. The logical name (LN) is used to access the COSEM object attributes and methods. Basically, the data exchange is performed between a client and a server through a service request and a service response. The policy engine we propose focuses on the client-server model because the DLMS/COSEM protocol is fundamentally a connection-oriented communication.

2.3 AMI Security

Smart meters have been shown to have vulnerabilities that may be exploited to infect them with malware and then “weaponize” them to spread malware [7] to other network elements in the AMI. Another major security concern for AMI is the so-called “off-switch”. This functionality enables remote control over the smart meters so that the utility might enforce load-shedding or be able to cut off power to defaulters. For an attacker, however, the off-switch opens possibilities of hijacking the grid, interrupting supply, and causing widespread chaos. Several attack scenarios based on the off-switch have been proposed in [3]. The work also highlights threats associated with another capability of the smart meters, i.e., remote software and firmware upgrades. Remote upgrade capability is absolutely necessary for future proofing because manual upgrades to millions of meters every time a new vulnerability is discovered or fixed may be prohibitively costly. On the downside, if a hacker can assume control over remote upgrades, then the hacker can infect the meters with bugs, or make them unresponsive. Concerns about remote disconnect capability have also been expressed in [8], which presents an attack tree to achieve a targeted disconnect of electrical service. To ensure that the smart

grid is well protected against security breaches, a sustained and concerted effort towards extensive and in-depth security and vulnerability analysis is needed. This is a huge endeavor because the smart grid is characterized by complex interactions of a large variety of devices and underlying technologies. At the same time, the smart grid security is also crucial to the success of the smart grid since it can streamline measures that must be taken in order to ensure stronger protection, detection, response, and recovery against a threat base that is still evolving.

Not only the smart meters are shown to be vulnerable, but also the infrastructure has been exposed to many security problems involving privacy, integrity, and authentication of metering data [9], [10], [11]. The smart grid is vulnerable to physical attack, data attack, and attack to network availability, and privacy [11], [12], [13]. False data can be injected to affect state estimation [14], [15]. The risk of anyone with sufficient technical skill to monitor meters for energy consumption patterns in entire neighborhoods introduces security and privacy concerns into automatic meter reading [10]. Researchers have proposed jamming-based defenses against spoofing attacks and privacy [10]. Weining Yang [12] has proposed a stepping approach for battery privacy algorithms to hide appliance loads from smart meters. The stepping approach was found to handle load peaks better while disclosing little information as defined by mutual-information metrics to evaluate the privacy of different algorithms.

3. AMI Vulnerability Assessment

3.1 Threat of DDoS

In this chapter, we present an in-depth study of the vulnerabilities in AMI to cyber-attacks. We point out vulnerabilities in ANSI C12.22, the protocol specification for interfacing smart meters with data networks. We also show how these vulnerabilities can be used to launch denial of service (DoS) attacks on the AMI and then propose potential solutions.

A distributed denial of service attack happens when a number of malicious network elements collude with each other to direct a large amount of traffic towards a common target victim. The intention is to exhaust the victim's resources while it tries to accept and process the large volume of traffic. At this point, the victim is unavailable to function properly and refuses to accept even legitimate traffic. In particular, a bandwidth depletion DoS attack targets the victim system's network bandwidth. The attack can be achieved by flooding the victim with huge amounts of traffic thus not leaving enough for legitimate users. Another way to launch a bandwidth depletion attack is via traffic amplification where attackers send messages to a broadcast IP address that solicit replies directed towards the victim.

For the Internet community, DDoS attacks are neither new nor uncommon. As smart grids and AMIs interface with IP networks in hopes of reaping benefits of higher speeds and sophisticated technologies, a growing concern is that AMI may simultaneously be inheriting the same vulnerabilities as seen in the IP networks. In the next section, we identify DoS scenarios that exploit vulnerabilities in ANSI C12.22 services.

3.2 C12.22 Vulnerabilities

3.2.1 Trace Service Vulnerability

ANSI C12.22 provides a "trace service" to find out the route that a C12.22 message traverses. The trace service is essentially a list of relays that forwarded the message on the way towards its destination. The trace can be obtained by making the message make one round-trip to the destination and back while collecting information about the route. Now we explain how this is done in the C12.22 trace service. Suppose a C12.22 node with a unique ID called ApTitle x wants to trace the route to another C12.22 node with ApTitle y . The sender initiates a trace service with the Calling ApTitle (sender ID) set to x and the Called ApTitle (destination ID) set to y .

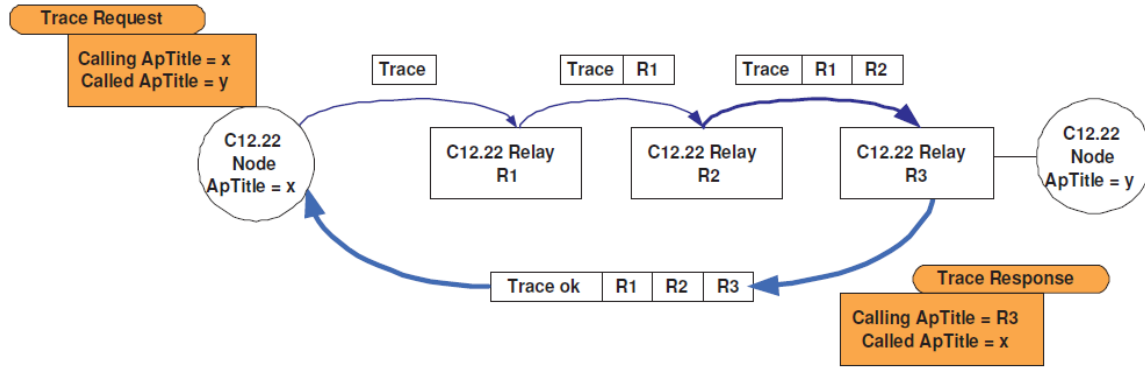


Figure 2 Graphical representation of the C12.22 trace service. For an attack, the attacker sending the trace request will set the Calling ApTitle to that of the victim node and the responses will all be routed to that ApTitle.

When a C12.22 relay receives the trace service request, it extracts the Called ApTitle and searches for it in its routing table. If a match is found, the relay forwards this message to the next-hop according to the routing entry. Before forwarding, the relay also adds its own ApTitle in the message to form a list of relays that forward this message. Every subsequent relay also appends its own ApTitle to this list before forwarding the request further. The process is shown in Figure 2 where relays R1 and R2 append their own ApTitles to the trace service request before forwarding it. When the trace request reaches a C12.22 relay that has the target node (whose ApTitle matches with the Called ApTitle) as a neighbor, this relay is required to send a trace service response. In Figure 2, R3 is such a relay since it has node y as its neighbor. For sending the response, relay R3 creates a trace response message, copies the list of relays from the request and appends its own ApTitle to the response, sets the Called ApTitle to the ApTitle of the originator of trace request, i.e., x, and the Calling ApTitle to its own, and sends out the response. Note that the size of the trace service messages increases as they traverse the network since all the relays add their own ApTitles to the message. For a long route, especially in a dense mesh network, the messages can grow fairly large because of multiple hops. The large-size messages can become problematic since applications may react unpredictably to messages of unexpected sizes.

To see how this service can be vulnerable to DDoS attacks, we present an attack scenario using Figure 2. Suppose the C12.22 node that sends out trace request x is the attacker and wants to attack another C12.22 node whose ApTitle is z. When the sender sends the trace request, instead of writing its own ApTitle in the Calling ApTitle field, it sets this to z, thus spoofing z's identity. When a relay like R3 sends trace response, it copies the Calling ApTitle from the trace request and sets it as the destination of the response. In the attack scenario, R3 is tricked into sending the response to z instead of x. When

colluding nodes request spoofed requests from the victim, all the relays route their responses to the victim node z, and the victim's network resources are exhausted resulting in DDoS.

3.2.2 Resolve Service Vulnerability

ANSI C12.22 also provides a “resolve service” to facilitate node discovery and retrieve the native network address of a C12.22 relay or node. The native address is used to communicate directly with other C12.22 nodes on the same native network or LAN. The service is defined by two messages: A “resolve request” and a “resolve response”. The resolve request contains the ApTitle of the C12.22 node for which the native address is requested, while the resolve response contains the required address. The C12.22 node that wants to retrieve the address of another node sends out a resolve request and includes ApTitle of the requested node as “Called ApTitle” and its own as the “Calling ApTitle”.

According to ANSI C12.22, every relay capable of forwarding information to this Called ApTitle should return a resolve response to the node identified by the Calling ApTitle. The use of this service is illustrated in Figure 3. Suppose the node with ApTitle x wants to discover the node with ApTitle y. x sends out a resolve request with Calling ApTitle (sender id) set to x and Called ApTitle (destination id) set to y. When the request reaches a relay like r which knows y, r sends a resolve response to x containing the native address of y. The address can be used by x for future communication with y. The resolve service can be used to find addresses of C12.22 master relays and also to retrieve the IP address of C12.22 IP relays that provide a route out of the C12.22 network segment.

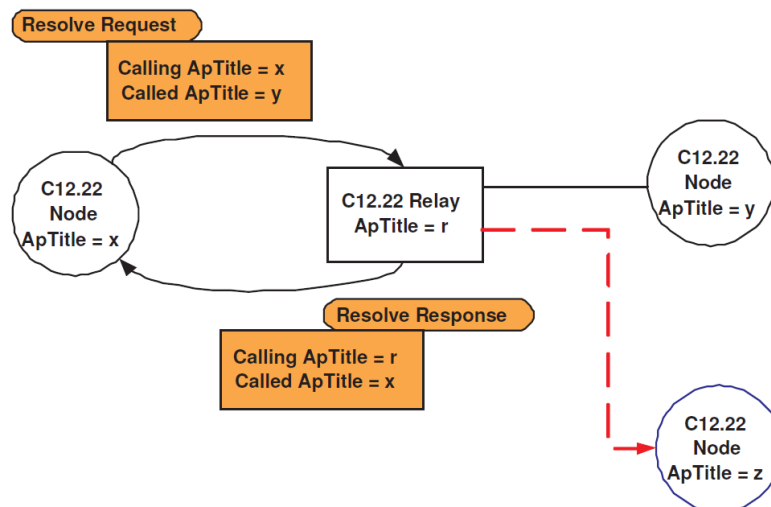


Figure 3 An illustration of the C12.22 resolve service.

We now explain how this service can be misused to launch a DDoS attack. Suppose a colluding group of compromised C12.22 nodes broadcast resolve requests for a certain C12.22 relay. Then, according to the standard's requirement, every C12.22 relay capable of forwarding to that node will return a response. If the malicious nodes collude and make it look like the requests are coming from the victim, then all the responses will be returned to the victim. This attack is further exacerbated from the victim's perspective because a single broadcasted resolve request may solicit responses from several relays hence amplifying the attack traffic. Therefore this attack is particularly tempting for attackers since they do not have to inject all the attack traffic themselves, the broadcast functionality does it for them.

Figure 4 shows a graphical illustration of an attack based on the resolve service. Attacking nodes spoof the identity of the victim and broadcast resolve requests to ask for the network address of some random node. Every relay capable of forwarding to that node will send a resolve response to the victim. If the attackers send resolve requests for a popular node, e.g., a master relay that all relays are aware of, the attackers can maximize the number of responses and make the attack very effective.

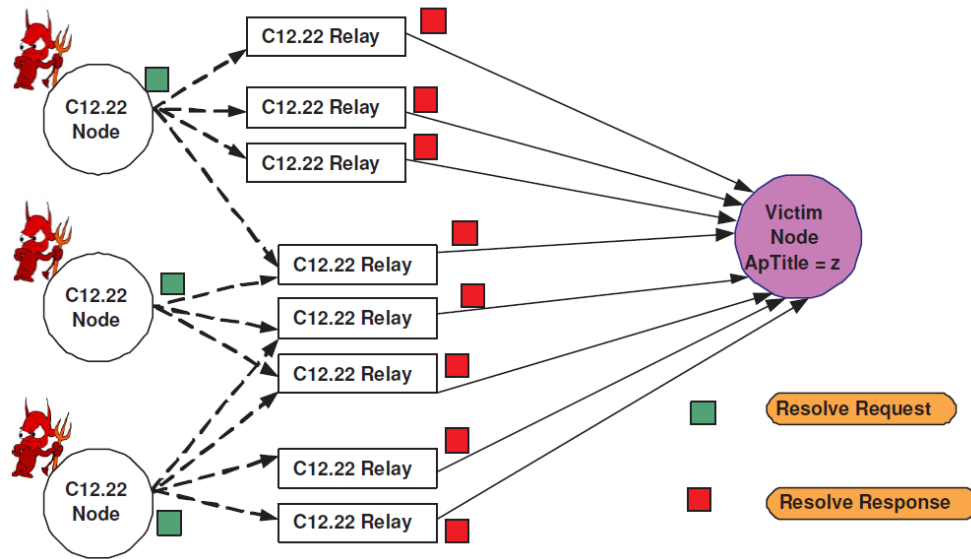


Figure 4 An amplification DDoS attack using the C12.22 resolve service.

3.2.3 Urgent Traffic Vulnerability

C12.22 provides for a way to give preferential treatment to critical messages such that these messages can get to the central office as soon as possible. The idea behind this provision is that certain messages, e.g., alert messages sent by smart meters when they detect power outages, are more important than others. If these messages are delayed in the network, the utility might not be able to take any recovery

or back-up measures in time. To provide priority for such messages, ANSI C12.22 provides a 1-bit indicator named URGENT. The protocol then requires that messages with the URGENT bit set to 1 must be forwarded by all relays and acted upon by the final destination urgently (with high priority). This feature can be easily attacked by potential attackers who always mark their messages as urgent and steal the resources required by legitimate traffic. If these messages are designed to come together at a particular node on the route to the central office, this node could start dropping legitimate traffic not marked as URGENT.

The attacks mentioned above utilize two basic techniques: source spoofing and sending messages via several relays. We claim that both of these actions are possible and in fact the work in [8] uses a meter-spoof program to demonstrate energy fraud. Similarly, sending messages through several relays can also be done without violating the ANSI C12.22 specification since the C12.22 specification allows a node to register with several different relays [4].

3.3 Proposed Solutions

The vulnerabilities identified above can be masked with careful design, planning, and implementation. In the case of the trace service, for example, the priority of trace messages may be kept low so that other network traffic does not suffer from delays and packet drops due to trace requests. Another approach to dealing with these attacks is to devise efficient measures of detecting and preempting the attack before it can cause significant damage. We try to characterize features of the attack traffic so the malicious traffic may be distinguished from legitimate traffic. Both the trace service attack and the resolve service attack contain at least one invariant: the victim's identity. All the traffic in both attacks must contain the identity of the victim. This may be used by the network to distinguish attack traffic from legitimate traffic. The network operators can then put limits on the number of trace or resolve requests that may be acceptable from a particular sending ApTitle. If the number of requests goes beyond this threshold over a certain period of time, then all subsequent requests may be dropped. The same threshold based provisioning argument can be made for an attack that exploits the URGENT bit functionality. The relays may be configured to give priority to URGENT traffic, but at the same time, they should not delay or drop other traffic beyond a certain threshold.

The rate limiting solutions mentioned above may not be effective when the attacker distributes its traffic over several relays that keep the individual rates below the threshold. To avoid such , we envision a much stronger solution, at the heart of which lies the fact that smart meters and hence C12.22 nodes

and relays communicate over the wireless channel. Wireless is a shared medium and when nodes in the AML mesh send messages, other nodes in the neighborhood can overhear them. A watchdog is a wireless node that receives messages overheard on the channel with the intention of monitoring its neighbors and traffic in the network. For example, a watchdog can gather information about the outgoing traffic rate of all its 1-hop neighbors. One of the popular arguments against wireless watchdogs is that promiscuous overhearing (and hence receiving) consumes significant energy. However, that concern is valid only for isolated, battery operated devices and not for smart meters that have direct sources of power. We will now discuss how watchdogs may be helpful in defending against the above-mentioned attacks.

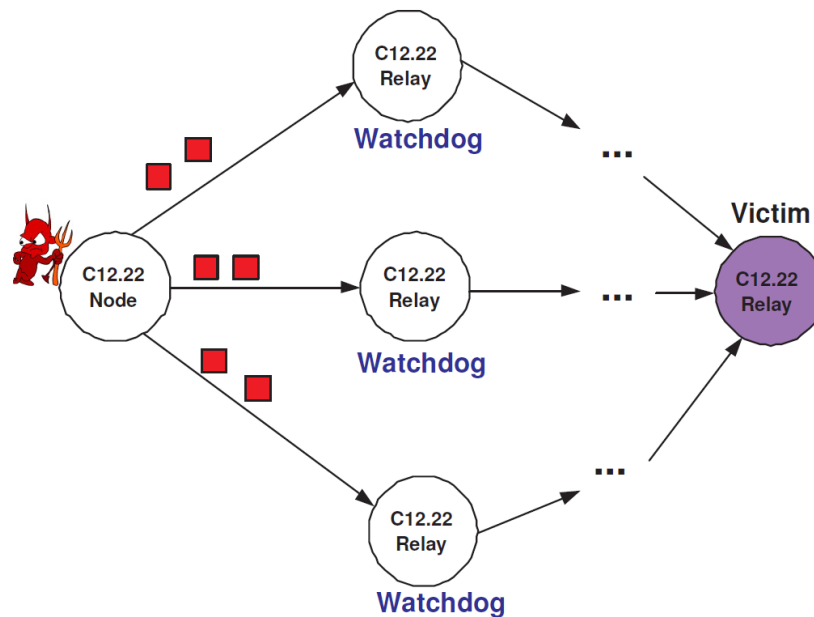


Figure 5 Wireless watchdogs against the URGENT-bit attack.

Consider Figure 5. Here, the attacker intelligently distributes its attack traffic and routes it via several relays so that at each individual relay, the rate of traffic always remains below the threshold. However, all this traffic may come together at the intended victim and may overwhelm it. Even if the victim is able to process and drop it, the attack traffic will have already consumed resources in the mesh by that time and may have caused delays to legitimate traffic. To see how watchdogs may help detect the attack, suppose the relay nodes were acting as watchdogs and maintaining a record of outgoing traffic from their neighbors. The attacker must be a neighbor to each of the relays to send its messages to them. Then, all three relays should be able to overhear all the messages from the attacker and can make a note of the total traffic originated by the attacker. By applying a threshold to this cumulative traffic, the

watchdogs can independently conclude that this node is originating attack traffic. Therefore, they can either provide a degraded quality of service to this node's traffic or even drop its messages. These watchdog-based solutions can be particularly effective and efficient since they do not even require any collaboration between the watchdogs. Each of them can make its own observations, and they should all still reach the same conclusion about attack detection. This solution can be very efficient in containing the attack to small neighborhoods.

Since the attacks we identify benefit greatly from ApTitle spoofing, as a final note, we now suggest how watchdogs can also be helpful for detection of spoofing. Since C12.22 nodes in the mesh network maintain a list of neighboring ApTitles, they can detect when they receive or overhear messages from a previously unknown ApTitle. There could be two reasons a message can originate from a new ApTitle:

- 1) A new node joined the mesh network and the new ApTitle belongs to this node.
- 2) A compromised node is originating messages by spoofing another node's ApTitle.

In either case, the nearby nodes (watchdogs) in the neighborhood will update their routing tables to indicate that this node is reachable in one hop. This information is also propagated through routing updates to the rest of the network, and so all traffic intended for this ApTitle will be routed to the node that most recently used the new ApTitle. In the first case above, there is no attack, so when intended traffic is routed to the new node, the network functions just as it should. However, for the second case, the attack traffic will be routed to the attacker instead of the victim, and thus the attacker will be overwhelmed by its own traffic rather than causing damage elsewhere.

The solutions we propose, while not perfect, are intended to be the starting point towards a larger effort to securing the grid. Protocol vulnerabilities can become hard to mask efficiently once large-scale development and deployment have been put in place. Therefore, we find it of utmost importance to investigate these issues in greater detail before their solutions become harder than they ought to be.

4. N-Gram Analysis for Malware

4.1 Traffic Type Classifier

A successful DDoS attack usually requires a large size botnet. Botnets sometimes compromise computers whose security defenses have been breached and control conceded to a third party. Each such compromised device, known as a "bot" is created when a computer is penetrated by software from a malware (malicious software) distribution. In this and following chapters, we will discuss solutions to prevent malware compromising AMI components.

Malware is typically propagated through a network by hiding itself into the incoming or outgoing traffic in a system. The main defense method against malware is to inspect whether the incoming or outgoing traffic includes malicious code. To accomplish this, we need a traffic classifier which can distinguish malicious code from legitimate data. In this section, we make use of FilePrint [16], which is an existing solution for file type classification. The basic idea of FilePrint is that each file type has a distinctive distribution even though a file type identifier is missed or obfuscated.

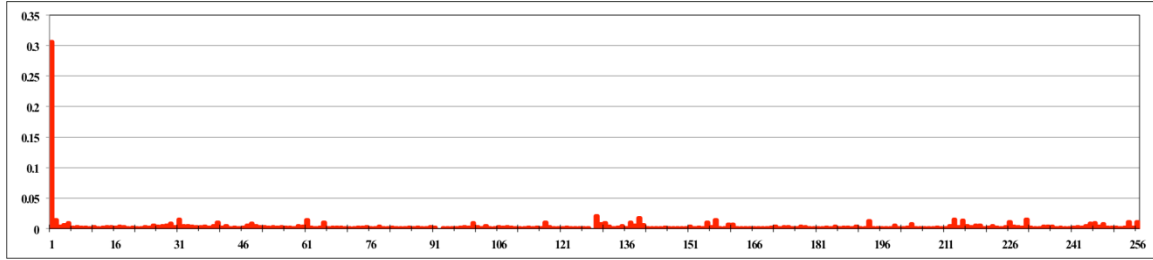
A standard compiler-generated executable has a standard file format with magic numbers to identify the file type. For example, PE executable has 4D 52 and ELF executable has 7F 45 4C 46 in the first part of the files. However, the pure signature of file types can be purposely missing and obfuscated. Therefore, file classifier investigates whether the traffic has executables or not by using the 1-gram distribution of executables as presented in [16]. In this work, we use a similar method to identify AMI network traffic instead of files.

4.2 Evaluation

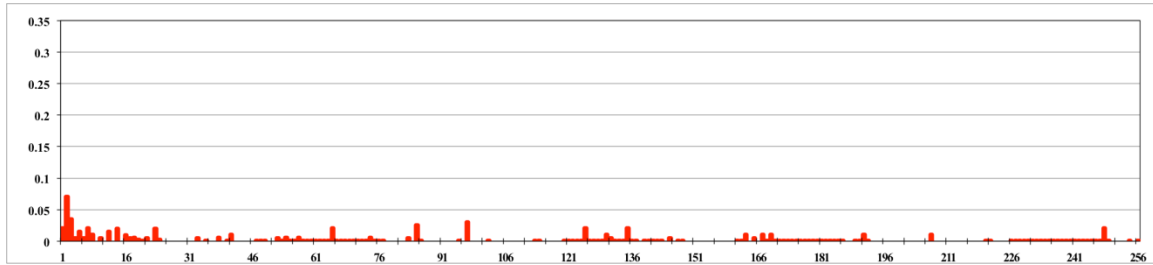
To make 1-gram distribution for malware, we collected 300 malware samples in the wild. Among the 300 samples, 200 malware samples were used to build a model of 1-gram distribution as a training set. The remaining 100 malware samples were compared with the test model. The collected malware was in an executable format.

Figure 6 demonstrates that the 1-gram distribution of metering data is different from that of malware. We derived the 1-gram distribution of byte values in the first 1000 bytes of each file as in FilePrint. It is shown that the distributions were quite noticeably different from each other. Specifically, 30% of malware distribution is gathered in the first byte (00h); however, only 7% is located in the first byte for

metering data. In addition, the metering data has non-zero distributions in only 40 different bytes, but the distribution of malware is spread through 256 different bytes from 00000000 to 11111111(00h to FFh). The experiment validated that the 1-gram distribution of malware, unlike that of metering data, serves as a good identifier to check a policy.



(a) One Malware Sample



(b) One Meter Data Sample

Figure 6 The 1-gram distribution for malware and metering data. Y-axis is a frequency of bytes in the file in the range of 0 to 1. The file is truncated in 1000 bytes. X-axis is a byte value from 0 to 256.

With the real metering data and the collected malware samples, we evaluated a false positive for correct metering data to be accidentally recognized as malware. Table 1 shows the result of false positive rate. Table 2 shows the false negative rate. K is the number of unique models built to represent a malware executable class. Unique training models were generated by K-means clustering and Manhattan distance as described [16].

The truncation indicates the fixed portion for the first part of a file when computing a byte distribution. Dissimilarity is estimated by Mahalanobis distance in [16] to compare a test file with the training models. The dissimilarity values ranged from 0 to 3. We set a threshold of 1 to decide whether metering data was matched with the training models of the malware samples. As a result, the designed policy rule with n-gram analysis showed less than 2% false positives. However, the tables indicated that the size of metering data was generally less than 1 KB. Therefore, the policy rule with n-gram analysis identifies malware in metering data with around 1% false positive rate. However, the n-gram analysis shows high

false negative rates, which means a large number of malware files cannot be detected by the traffic classifier.

Table 1 False positives of n-gram analysis

Size	50 bytes	100 bytes	200 bytes	500 bytes	1 KB	2 KB
K=1	0.00%	0.00%	0.00%	0.00%	0.00%	1.00%
K=2	0.00%	0.00%	0.00%	0.00%	1.00%	1.00%
K=5	0.00%	0.00%	0.00%	0.00%	1.00%	1.00%
k=10	0.00%	0.00%	0.00%	0.00%	1.00%	1.00%
k=20	0.00%	0.00%	0.00%	1.00%	1.00%	2.00%

Table 2 False negatives of n-gram analysis

Size	50 bytes	100 bytes	200 bytes	500 bytes	1 KB	2 KB
K=1	13.50%	9.00%	24.00%	13.50%	10.00%	19.50%
K=2	10.50%	5.00%	12.50%	9.50%	5.50%	6.00%
K=5	9.50%	4.00%	11.50%	9.50%	5.00%	3.50%
k=10	9.50%	4.00%	7.50%	8.50%	5.00%	3.50%
k=20	8.00%	4.00%	3.50%	3.50%	3.00%	1.00%

The experimental results show that n-gram analysis works effectively for detecting some malwares but not all of them due to the high false negative rates. There are also some drawbacks of using n-gram analysis for detecting malware, which make it impractical to implement in AMI system. One important assumption of n-gram analysis is that the malware is carried in its native form. Unfortunately, the attacker can easily encrypt or compress the malware in order to alter the byte distribution. Note that the authors of FilePrint [16] claim that the head of the file is the most important part for file type identification. However, packets might arrive out of order in the network, and the attack might permute the bytes purposely to hide that information. Another big challenge we face is building models for malicious and legitimate traffic. The model trained on one data set might not work for all other traffic of the same type. These limitations motivate us to propose a more effective method to detect malware propagation in AMI networks.

5. Policy Engine Design

5.1 System Architecture

The proposed policy engine is layered between applications and the protocol stack of DLMS/COSEM as shown in Figure 7. The policy engine scrutinizes incoming and outgoing traffic on the devices before the applications running on the devices accept the control commands and data, and before the data penetrates the protocol stack or the supporting layers. As shown in Figure 8, the policy engine consists of three systems to detect non-metering data and to prevent the traffic from carrying malware. The following sections will describe the proposed system in detail.

- Data collection system: This system profiles invariant features during data exchange in the system related to the standard protocol.
- Policy rule monitoring system: This system defines a variety of policy rules according to the standard protocol and statistical features. By monitoring communications, it examines incoming and outgoing traffic semantically and statistically.
- Logging system: When the policy engine detects malicious code in the traffic, this system makes a record for further analysis.

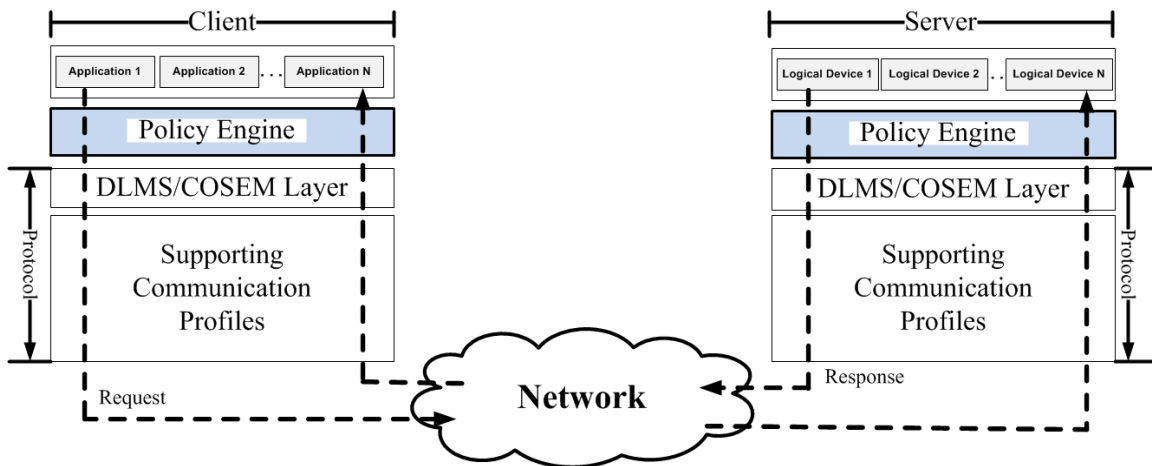


Figure 7 The placement of the policy engine in the AMI system.

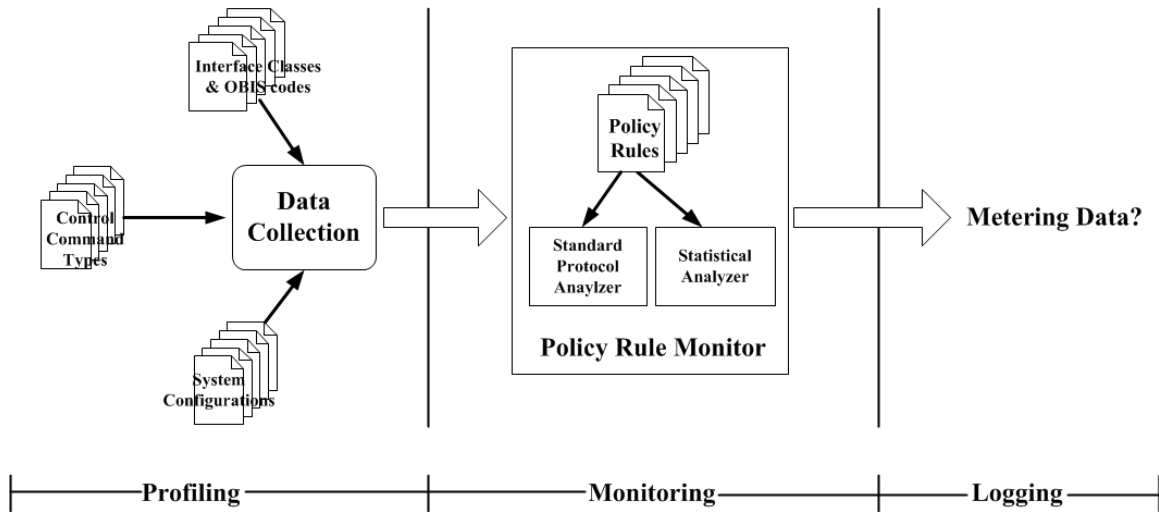


Figure 8 Architecture of the proposed policy engine.

5.2 Data Collection

The policy engine first collects invariant information from the systems, including system configuration data, a list of OBIS codes and COSEM interface classes, and control command types. The list of OBIS codes and COSEM interface classes is obtained from the DLMS user association. Based on these standard documents, the proposed policy engine first profiles a subset of interface classes implemented on a particular system during installation or at run-time. The control command types provide invariant information for successful communications: GET/SET/ACTION/EventNotification services for LN (logical name) referencing and READ/WRITE/InformationReport for SN (short name) referencing. Finally, the system configuration data, such as a device name, a device ID, an IP address, a time zone, measurement unit, or the like, are set at the time of installation.

When AMI systems are installed, the configuration information for the smart meter and its internal structure should be securely set up to ensure subsequent successful communications. Such information may also include ready-to-use tariff schedules and installation meter-specific parameters. The object list available in the server and the access rights to the objects' attributes and methods can be retrieved by reading the second attribute value of the application association during communication or during installation.

5.3 Policy Rule Monitoring System

The policy rule monitoring system investigates incoming and outgoing traffic in real time by using various policy rules based on analysis as shown in Table 3. The system has two components: standard

protocol analysis and statistical analysis. The standard protocol analysis examines the traffic according to the DLMS/COSEM application protocol. Different kinds of policy rules are defined to check whether the traffic conforms to the specifications of the standard application protocol. In addition to this, the statistical analysis inspects whether the traffic contains the obvious features of malware; most malware uses packing and encryption; malware is executable. By using entropy, pattern, and signature analysis, the policy engine monitors these three traits of metering data that are statistically different from malware with byte streams. The detailed methods for each component will be described in the following sections.

Table 3 Policy rule monitoring system

Components	Analysis for Policy Rules
Standard protocol analysis	Syntactic policy rules Semantic policy rules Access policy rules Communications policy rules
Statistical analysis	Entropy analysis Pattern analysis Signature analysis

5.4 Logging System

The policy engine allows traffic to pass if it satisfies all of the policy rules. Otherwise, when rules failures suggest anomalies, the engine records the message for further investigation. The policy engine logs the number of violated or specific rules, and it dumps the whole traffic to files. The logs and files can be forwarded to another intrusion detection system. The saved logging information can be forwarded into an external intrusion detection system that might be integrated with the proposed system.

6. Standard Protocol Analysis

6.1 Overview

The standard protocol analysis is organized according to four different sets of policy rules: syntactic policy rules, semantic policy rules, access policy rules, and communication policy rules. Each policy rule related to the DLMS/COSEM protocol plays a special role in identifying abnormal data in an AMI system. Depending on control command types, the traffic will be investigated by using a subset of policy rules that are dynamically selected from the pool of policy rules. The policy rules are extendable when different communication environments or new data objects are built by individual manufacturers, but the same rules apply in all cases.

6.2 Policy Rules

6.2.1 Syntactic Policy Rules

Syntactic policy rules describe the proper syntax of the DLMS/COSEM protocol based on specific packet formats and unique identifiers. Syntactic policy rules verify static and invariant information during communications between a server and a client. The policy rules basically examine the invariant data, such as Class ID, OBIS codes, system configuration data, and the index of attributes and methods for the installed data objects based on the information from the DLMS UA. In addition, these rules investigate the packet format of the DLMS/COSEM protocol.

6.2.2 Semantic Policy Rules

Semantic policy rules define the proper range of data objects and the proper operation of the DLMS/COSEM protocol. The policy rules verify the semantics of the traffic during communication, such as the range of data types for each object, attribute values, and mapping of requests and responses for services. Each data object has static and dynamic information that can be accessed by the index of attributes. The static information is invariant and can be verified by the syntactic policy rules. However, the dynamic information is periodically updated, so the information needs to be checked using the semantic policy rules. To check the dynamic information, the policy engine should set up a threshold criterion. For example, when a client requests a clock object to indicate the current time, the current system time should be checked with a predefined offset as a threshold. The policy rules should set a threshold heuristically or systematically during device installation time.

6.2.3 Access Policy Rules

Access policy rules control a set of constraints for smart meters and define the correct reference rules for each data object. They describe valid access control rules for an object, its attributes, and methods. The access policy rules examine access privileges and access rights for the defined data objects, attributes, and methods. For example, to modify the values of attributes, the policy engine checks whether the request action service for LN referencing has valid access rights with written permission. In addition, the static information about the attributes is preserved at runtime because it is invariant information. For example, the first attribute of each object is a logical name that cannot be changed.

6.2.4 Communication Policy Rules

The communications policy rules describe the proper sequence and operation of communication between a server and a client. These policy rules verify communication information that is necessary for data exchange, such as IP addresses, port numbers, and application addresses. For example, the number of connections should not exceed any limitations on communication availability. The sequence of communications should follow a correct state machine as defined in the COSEM protocol, IEC 62056-53. The DLMS application protocol is a connection-oriented protocol that allows client and server to use any service once they are associated. To exchange application data, a client AP and a server AP establish a connection (association), transfer data, and then the connection terminates at the end of the data exchange. These communication rules should verify the associated connection with the correct port numbers, the sequence of communications, and the special addresses for special application processes. For example, the request service from the client should be accompanied by the corresponding response service from the server. The application address of the public client is reserved with 0x10 to discover the structure of an unknown meter. The management logical device has a special application address 0x01.

7. Entropy Analysis

7.1 Entropy of Network Traffic

Like the tool Bintropy [17], the policy engine computes the entropy of a stream of n bytes as:

$$H(x) = - \sum_{i=1}^n p(i) \log_2 p(i)$$

where $p(i)$ is the relative frequency of the value of the i^{th} byte in the series of n bytes. The entropy value ranges from 0 to 8, with larger values indicating more internal disorder. Good encryption masks internal order of the plain text, and so encrypted text tends to have high entropy. Evidence reported in [17] suggests that encrypted files have entropy close to 7, whereas unencrypted files have entropy that is much lower. Our entropy test rule raises an alarm if a payload's entropy exceeds 6.67.

When compared with the Bintropy, the metering data have very low entropy, because the metering data do not make use of packing tools and encryption on devices. However, attackers might use packing or encryption techniques to defeat against signature-based malware detection systems. Therefore, using entropy information about packing and encryption is one obvious way to detect traffic with malware.

7.2 Evaluation

Figure 9 illustrates the measured entropies of encrypted and unencrypted ARM files of size 2 KB or less, and measured entropies of unencrypted metering data packets with sizes in the same range. There is a clear distinction between encrypted and unencrypted data; indeed, on a set of nearly 100 ARM files of sizes up to 300 KB the entropies of encrypted binaries (using AES-256) are all very close to the maximum value, 8. The distinction is not so clear between unencrypted binaries and unencrypted metering data. With an entropy threshold of 6.677, on this data set we achieve perfect resolution—no false negatives and no false positives.

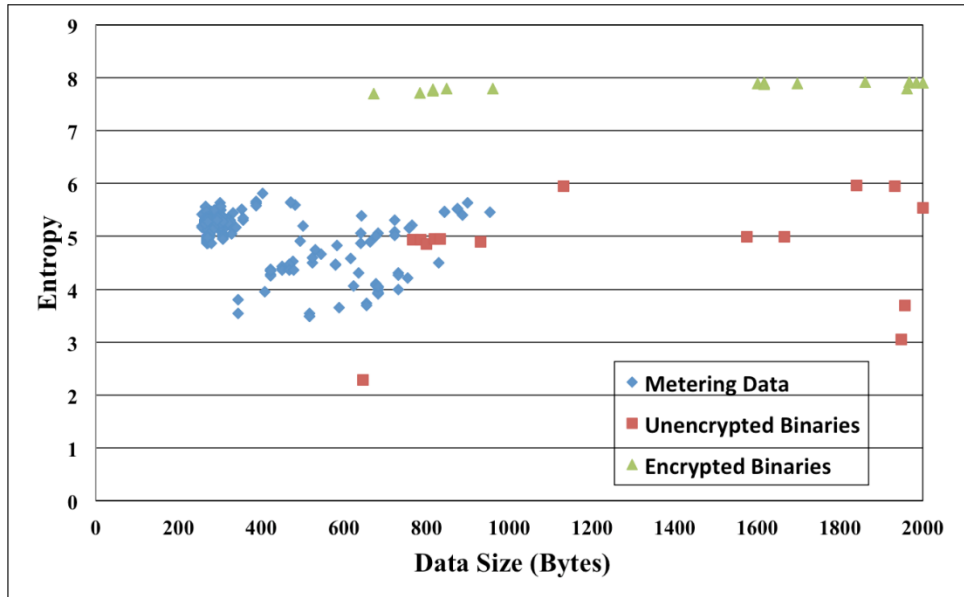


Figure 9 Entropy encrypted ARM codes, unencrypted ARM codes, and unencrypted metering data.

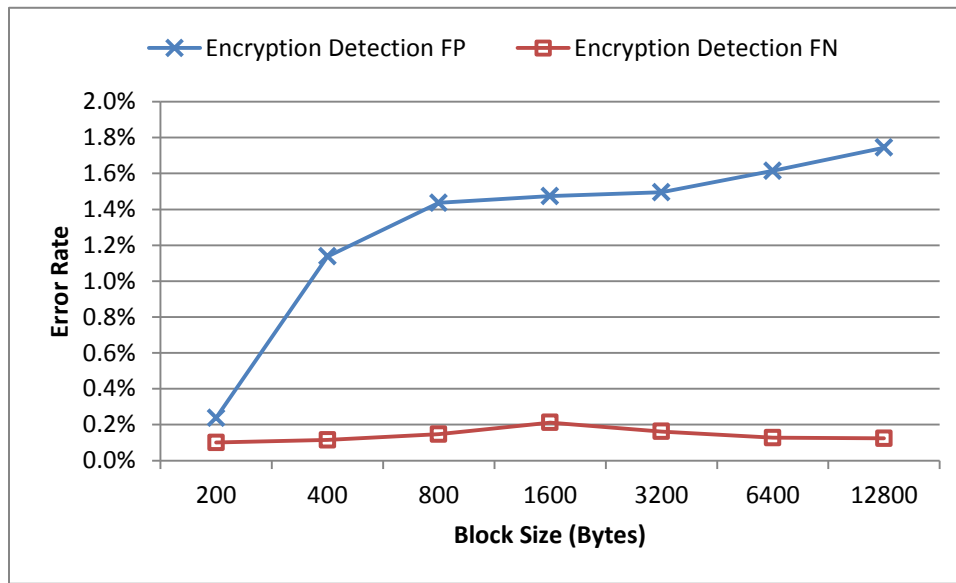


Figure 10 Encrypted message detection error rates.

Figure 10 shows the error probabilities of entropy analysis detection. We vary the size of the packets received by the policy engine, then test the fraction of native packets that have been recognized as encrypted packets (false positive), and the fraction of encrypted packets that have been recognized as native packets (false negative). From the figure we can see that both error rates are low, which means the entropy analysis has high accuracy in identifying encrypted packets.

The entropy tests we showed in Figure 9 and Figure 10 are the entropy of byte stream of the whole file. We have also done entropy analysis on each 4-bit or 8-bit segment within an ARM instruction. Figure 11 shows the average entropy of each 4-bit segment of ARM instruction (8 segments for one instruction). Since the four bits give 16 distinct values, the entropy value ranges from 0 to 4. Figure 12 shows the average entropy of each 8-bit segment (one byte) in the ARM instruction. The entropy value ranges from 0 to 8.

We notice that the seventh segment in Figure 11 and the fourth segment in Figure 12 of the ARM instruction give much lower entropy values. Since entropy is a measure of the uncertainty of information, the low value suggests some patterns in that field. This observation motivates us to find patterns in ARM code, which will be discussed in the next chapter.

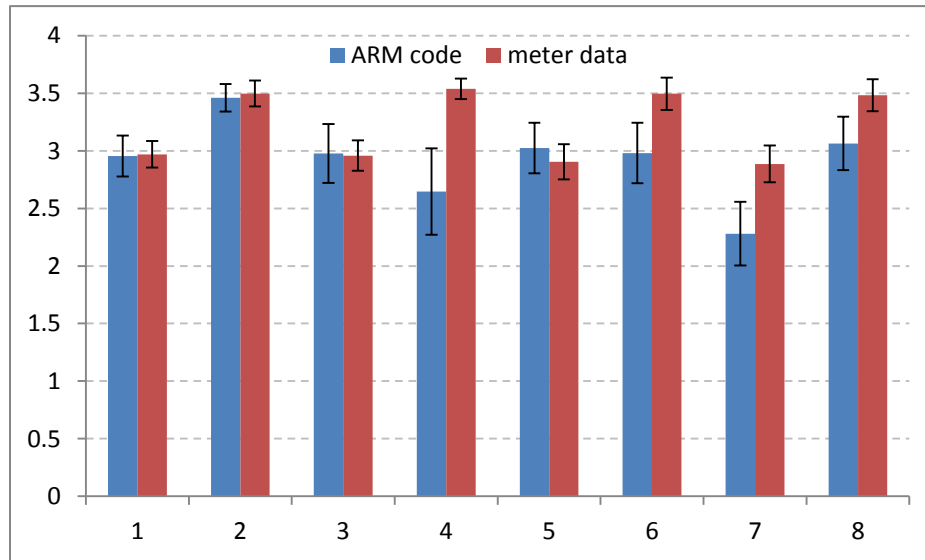


Figure 11 Four-bit segment entropy test.

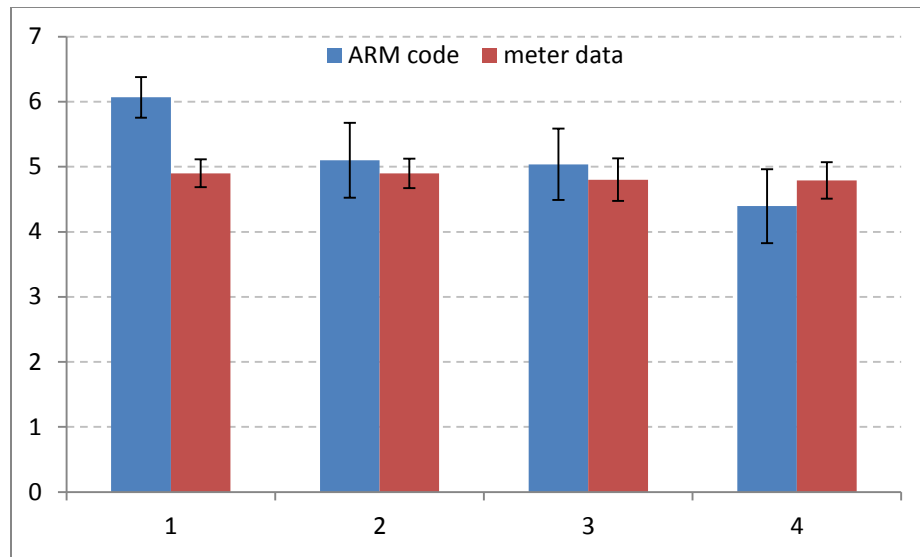


Figure 12 Eight-bit (one byte) segment entropy test.

8. Pattern Analysis

8.1 Patterns in ARM Code

Binary executables exhibit considerable structure, with certain patterns appearing frequently. For example, commands that move data from one register to another or instructions that clear registers are frequent. While the specific patterns will depend on the CPU type, there is, we believe, a solid principle at play.

For the purposes of demonstration, we studied a large number of programs compiled to run on the ARM processor. Those ARM programs are chosen as many AMI components are built around them. We observed visually, and confirmed empirically, that almost half of the instructions have a particular 4-bit field with value e in hexadecimal. These are the leading four bits of the instruction, and e is a code that indicates the instruction is unconditional, i.e., its execution is not dependent on the value in any status register. It seems highly unlikely that metering data should exhibit this same characteristic, so the condition field frequency test has promise.

The ARM processor has what is for us the attractive feature that all instructions are four bytes long. Still, given a block of data that might be an executable, we cannot assume that the first byte is necessarily the first byte of an instruction. However, assuming that the block is some contiguous sequence of bytes from an ARM executable, we know that for some offset the sequence of 32-bit ensembles that start at the offset form a sequence of instructions, and we can pull the condition code from each. Thus, for every offset, we compute the relative fraction of supposed condition codes that have value e , and if one of those relative fractions is very high, we raise an alert.

8.2 Evaluation

One problem the e -code test must face is finding the alignment of potential instructions in the sequence. For simplicity we supposed that the instruction would be aligned within the payload with a resolution of 4-bits, meaning that there are eight possible offsets where an instruction might begin. Expanding the analysis to the full set of 32 possible offsets is straightforward. Figure 13 shows the results of computing for each slide the fraction of e codes, over a data set of 50 program segments, of full size, for both ARM binaries and metering data. The graph plots standard deviations for the measurements. We see that one slice stands out as having very high e -code counts. The significant

variance observed there is due to the presence of data blocks in the ARM codes that do not contain instructions, and hence should not be expected to contain e codes.

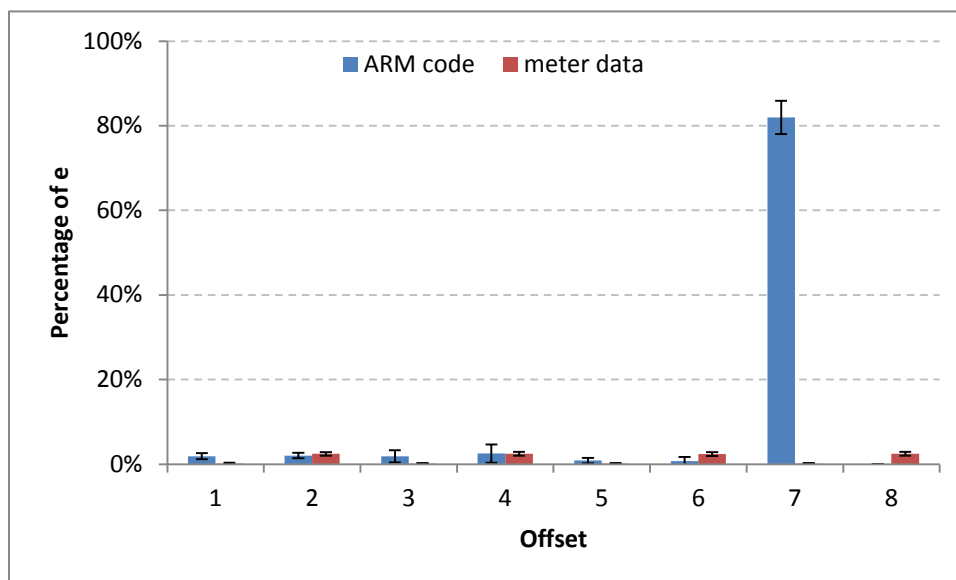


Figure 13 Percentage of 4-bit codes equal to e , by slice.

Next we tested the accuracy of the e -code test on a set of 85 ARM executables. Our experiment chooses block sizes of S : 100, 200, 400, 800, 1600, 3200, 6400, and 12800 bytes. For each block size S , a file is partitioned into contiguous blocks of S bytes, but discards those blocks that do not contain any executable code. The logic here is that while a message carrying a piece of an ARM data segment may easily slip past the e -code test, the propagation is not a threat until executable code is included.

A block is judged to carry executable code if some slice exhibits 20% or more of the e character. With our blocks selected to contain ARM code, for each block size S we compute the fraction of blocks of size S where the e -code fails. For similar block sizes of metering data to which the e -code test is applied, we compute the fraction of blocks where the e -code test indicates this is an executable. The results are plotted in Figure 14. We observe remarkably good error rates, including false positive and false negative rates. Clearly the e -code test has excellent discrimination.

The e -code test that appears to be very effective for detecting ARM binaries has limitations. Colluding compromised applications could easily defeat it by permuting the bytes. The x86 code is more challenging because of its variable instruction lengths. However in preliminary related work, we have found that a set of tests that look for commonly occurring patterns in x86 binaries can likewise detect

the presence of x86 code. Here the fingerprint is when a large enough number of pattern searches all “see” evidence of the x86 code.

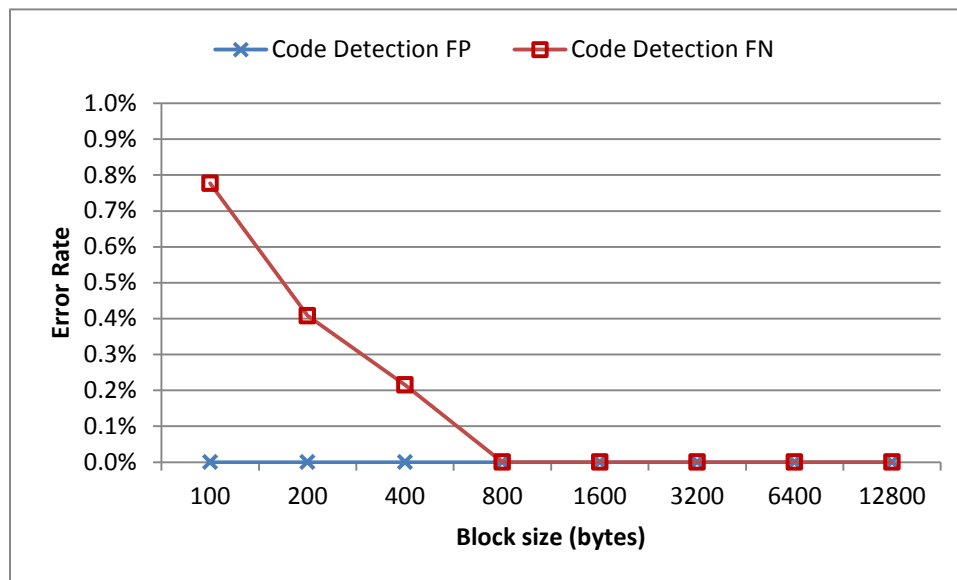


Figure 14 Accuracy of the e-code test.

9. Signature-Based Analysis

9.1 Overview of Signature-Based Analysis

The bit patterns of a specified field of an ARM instruction are effective in identifying native ARM code. However, bit-pattern recognition on ARM instruction faces two challenges. The first challenge is to find the start position of ARM instruction. To find the position, we must consider different ways in which the byte stream is aligned with respect to the beginning of the data packets. The second challenge is to deal with the case in which the attacker permutes the bytes of a packet in order to avoid detection.

The above concerns motivate us to explore the statistical characteristics of the ARM instruction bytes instead of finding the pattern in the particular field. If we are able to find some bit patterns in bytes of ARM code, the patterns might be useful in distinguishing ARM data from meter data. In this section, we introduce a method to find byte-level patterns as signatures that can be employed to identify ARM code. We also describe the detection algorithms used by our policy engine. The detection errors are evaluated based on real meter data and ARM executable files.

The goal of using signatures is to differentiate executable from meter data. There are two obvious ways to obtain the signatures. One is to look for patterns that occur significantly in ARM codes but are hardly seen in meter data. The other is to explore meter data patterns that have the low frequencies in ARM data. Since we try to catch the executable that was carried in the traffic, we focus on high frequency ARM patterns. In this work, we represent a signature by a string like `xx10xxx1` where the `x` character matches any bit.

9.2 Experimental Setup

We collected real metering data from the TCIPG [18] testbed. The metering data set is based on the C12.22 protocol, which is formatted in the form that the policy engine will see. We also obtained more than 100 ARM executables and libraries from an embedded board to represent the executable content carried in the network traffic. The data set is divided into two parts. One is a training data set, which is used to find signatures and define policies. The other set is used for evaluation purposes. The experiments are conducted with a stand-alone program module, which is later integrated into the policy engine. We also evaluated the performance overhead of the policy engine running on a single-core machine with 2 GB memory.

9.3 Signature Selection and Policy Design

Table 4 shows 10 selected signatures with high frequency of occurrence in ARM code and low frequency in meter data. We divide each ARM executable file and meter data file into 50-byte blocks and calculate their average frequencies and standard deviations.

Table 4 Selected signatures and thresholds

Signature	ARM-avg	ARM-SD	Meter-avg	Meter-SD	Threshold
<i>xx10xxx1</i>	0.190981	0.0506477	0.0163375	0.0180393	0.06
<i>x1x0xxx1</i>	0.192723	0.0513551	0.01431	0.0173437	0.06
<i>x110xxxx</i>	0.23595	0.0469141	0.058092	0.0157645	0.1
<i>1xx0xxx1</i>	0.203583	0.0570596	0.0237255	0.0140147	0.06
<i>1x1x0xxx</i>	0.258449	0.0678876	0.0815785	0.0239478	0.12
<i>1x10xxxx</i>	0.292047	0.0707724	0.065665	0.0198652	0.1
<i>11xxxxx1</i>	0.21666	0.062046	0.043394	0.0449878	0.08
<i>11xx0xxx</i>	0.210888	0.0509753	0.038962	0.0134522	0.08
<i>11x0xxxx</i>	0.240994	0.0491752	0.021727	0.0139706	0.06
<i>111xxxxx</i>	0.225502	0.0455125	0.002035	0.00706107	0.02

The goal is to use the significant difference of average signature-match frequencies to distinguish ARM packets and meter data packets. To this end, we need a policy to identify the ARM data effectively. The policy we used in this work is a simple threshold which classifies a packet as legitimate meter data with signature-match frequency lower than it and classifies a packet as malicious ARM data with frequency higher than it. The thresholds should be chosen carefully to reduce the error rates. As shown in Table 5, a large fraction of meter data packets of various sizes exhibit ARM data patterns in at least one of their bytes. Table 6 shows that there are ARM executable packets with no pattern in any of the bytes. The tables tell us that errors are expected for signature matching. These observations motivate us to find a good policy and design an effective signature-matching algorithm to minimize the error rates.

There are two errors involved in our policy engine. One is the false negative which classifies an ARM executable packet as meter data. The other is the false positive, which recognizes a meter data packet as ARM executable. If we define a high differentiating threshold, the false positive rate will be reduced while the false negative rate rises. A low threshold limits the false negatives but causes more false positives. Since control of false positives and false negatives is considered to be equally important, we choose the threshold that balances two error rates. Figure 15 illustrates the distributions of frequencies

of a signature in ARM executables and meter data. The optimal differentiating threshold is chosen such that $area_A$ (false negative probability) and $area_B$ (false positive probability) are equal. We have measured the frequencies of signatures and obtained the optimal threshold for each signature as shown in Table 4.

Table 5 Fraction of meter packets with at least one byte matching the signature

Signature	50 bytes	100 bytes	200 bytes	400 bytes	800 bytes
<i>xx10xxx1</i>	0.57792	0.859416	0.996495	1	1
<i>x1x0xxx1</i>	0.577056	0.846833	0.995911	1	1
<i>x110xxxx</i>	0.987037	0.993202	0.99854	1	1
<i>1xx0xxx1</i>	0.918119	0.991322	0.996203	1	1
<i>1x1x0xxx</i>	0.979116	0.987706	0.993867	1	1
<i>1x10xxxx</i>	0.983221	0.988863	0.995035	1	1
<i>11xxxxx1</i>	0.987397	0.993636	0.997956	1	1
<i>11xx0xxx</i>	0.981276	0.988574	0.995327	0.999398	1
<i>11x0xxxx</i>	0.861875	0.985826	0.998248	1	1
<i>111xxxxx</i>	0.091963	0.168933	0.260514	0.343769	0.411613

Table 6 Fraction of ARM packets with no pattern-match at all

Signature	50 bytes	100 bytes	200 bytes	400 bytes	800 bytes
<i>xx10xxx1</i>	0.007914	0.003544	0.001417	0.000566	0.000226
<i>x1x0xxx1</i>	0.007786	0.003941	0.001417	0.000453	0.000226
<i>x110xxxx</i>	0.002581	0.001077	0.00051	0.000339	0
<i>1xx0xxx1</i>	0.009417	0.004735	0.001983	0.000679	0.000226
<i>1x1x0xxx</i>	0.004638	0.002297	0.001077	0.000453	0.000226
<i>1x10xxxx</i>	0.002581	0.001389	0.00068	0.000339	0
<i>11xxxxx1</i>	0.006325	0.003601	0.00153	0.000113	0
<i>11xx0xxx</i>	0.00478	0.002608	0.001247	0.000339	0.000226
<i>11x0xxxx</i>	0.003134	0.001531	0.000737	0.000339	0
<i>111xxxxx</i>	0.004524	0.0019	0.000907	0.000339	0

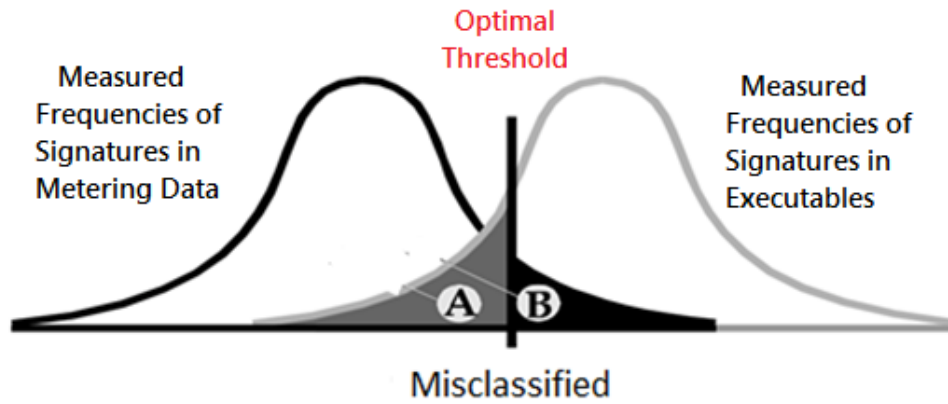


Figure 15 Optimal threshold to balance errors.

9.4 Error Analysis for Single Signature

Table 7 False negative rate of signatures

Signatures	Threshold	50 bytes	100 bytes	200 bytes	400 bytes	800 bytes
11x0xxxx	0.06	0.00735035	0.00464943	0.00224048	0.00149229	0
111xxxxx	0.02	0.00795249	0.00398522	0.00182557	0.00033162	0
1x10xxxx	0.1	0.0158427	0.00788742	0.00307028	0.00182391	0.00066203
x1x0xxx1	0.06	0.0151783	0.0084686	0.0047299	0.00198972	0
1xx0xxx1	0.06	0.0167563	0.00900826	0.0051448	0.00232134	0.00066203
xx10xxx1	0.06	0.0153028	0.0091328	0.00506182	0.00265296	0.00033102
x110xxxx	0.1	0.0195179	0.0091328	0.00448096	0.00232134	0.00033102
11xx0xxx	0.08	0.021262	0.0103367	0.00580865	0.00198972	0.00066203
1x1x0xxx	0.12	0.0239821	0.0145295	0.00970874	0.0049743	0.00264813
11xxxxx1	0.08	0.0227778	0.0123708	0.00697038	0.00265296	0.00099305

Table 8 False positive rate of signatures

Signatures	Threshold	50 bytes	100 bytes	200 bytes	400 bytes	800 bytes
11x0xxxx	0.06	0.041377	0.0112882	0.00403226	0	0
111xxxxx	0.02	0.0165508	0.0179283	0.0134409	0.00833333	0.00591716
1x10xxxx	0.1	0.0165508	0.00066401	0	0	0
x1x0xxx1	0.06	0.040053	0.0152722	0.0188172	0.0138889	0.00591716
1xx0xxx1	0.06	0.0675273	0.0205843	0.00268817	0	0
xx10xxx1	0.06	0.0589209	0.0278884	0.00268817	0	0
x110xxxx	0.1	0.00297915	0.00066401	0.00134409	0	0
11xx0xxx	0.08	0.00066203	0	0	0	0
1x1x0xxx	0.12	0.0665343	0.00664011	0	0	0
11xxxxx1	0.08	0.0301225	0.0146082	0.0120968	0.0222222	0.0473373

Table 7 reports the false negative rate for each signature that we have chosen. As we can see from the table, the error probabilities are affected by the packet size. A large packet adds more confidence for the signature to detect the executable. Similar to Table 7, Table 8 shows the probabilities of false positives. As shown in the tables, using any single signature limits the false positive rate under 0.07 and false negative rate under 0.03. However, the error probabilities are still too high to be used in practice.

9.5 Pattern Recognition Using Multiple Signatures

To reduce the error rates, we use multiple signatures instead of single signature for identifying executable patterns. We use a simple multi-signature voting scheme to identify the executable packets. If at least K out of N signatures recognize the packet (K votes) as executable packet, the packet will be flagged as malicious and dropped immediately.

9.5.1 Independent Signatures Modeling and Analysis

If the signatures we use are independent of each other, we will be able to compute the error probabilities with the following analytical model. Let us define the error probability of misclassifying a packet (size n) for N independent signatures as: $r_1(n), r_2(n) \dots r_N(n)$. Suppose we use N signatures to identify the type of a packet. If at least K out of N signatures misclassify the packet, we fail to identify the packet. Let α denote the set of N signatures, then the error probability of misclassifying a packet using multiple signatures is p :

$$p = \sum_{v \subseteq \alpha, \#v \geq K} \prod_v r_{v_i}(n) \prod_{w = \alpha - v} (1 - r_{w_j}(n))$$

Figure 16 and Figure 17 report false negative probabilities and false positive probabilities, respectively. We choose 10 signatures and 100 bytes as the packet size. The above model predicts the error probabilities of N -signature- K -vote schemes. The figures show that multiple signatures are useful in controlling false negatives but will generate more false positives. However, the signatures listed in Table 9 are not independent. Thus, we cannot get the significant drop in false negative rate shown in Figure 17.

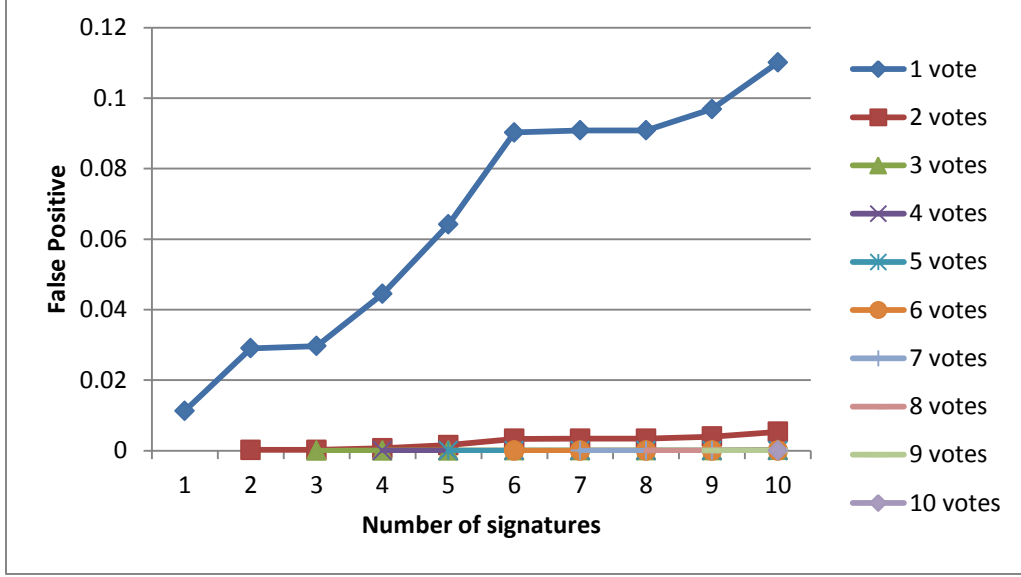


Figure 16 False positives of multiple independent signatures.

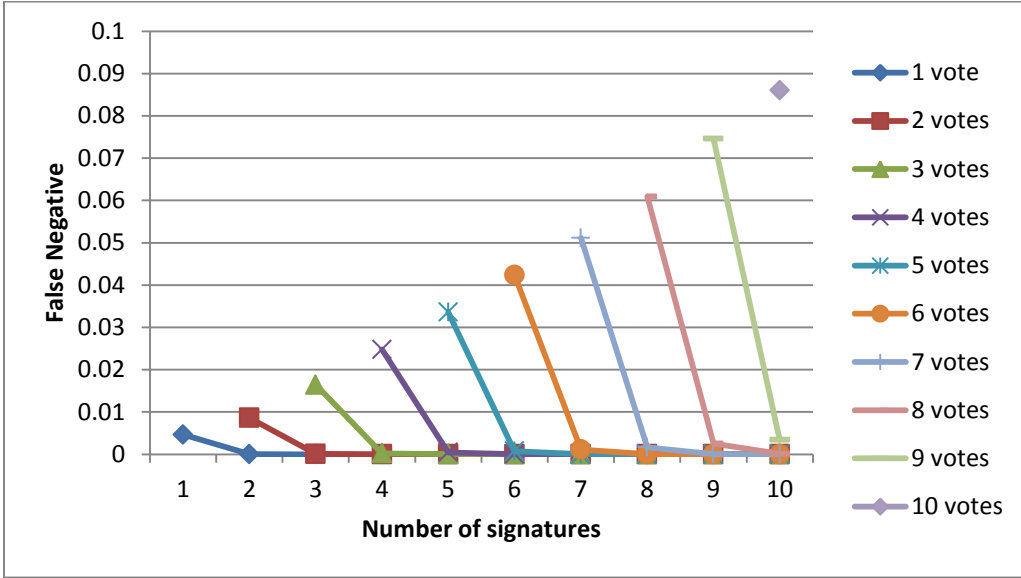


Figure 17 False negatives of multiple independent signatures.

9.5.2 Correlated Signatures Modeling and Analysis

Since the signatures we choose are not independent of each other, we study the joint distribution of using multiple signatures. Table 9 presents the joint distribution of using 10 signatures for 100-byte packets. The joint distributions for ARM data and meter data are given in Table 9. For example, $Prob\{012\}$ is the probability that the packet is recognized by signature-0, signature-1 and signature-2 but not recognized by any other signature. The zero probability cases are not shown in the table.

Table 9 Joint distributions of using ten signatures

ARM data		
<i>Prob{0}=4.15127e-005</i>	<i>Prob{0126}=0.000871767</i>	<i>Prob{0123456}=0.000415127</i>
<i>Prob{1}=0.00045664</i>	<i>Prob{0127}=4.15127e-005</i>	<i>Prob{0123458}=8.30254e-005</i>
<i>Prob{2}=0.000124538</i>	<i>Prob{0129}=4.15127e-005</i>	<i>Prob{0123459}=0.000166051</i>
<i>Prob{3}=4.15127e-005</i>	<i>Prob{0135}=4.15127e-005</i>	<i>Prob{0123569}=4.15127e-005</i>
<i>Prob{4}=8.30254e-005</i>	<i>Prob{0139}=4.15127e-005</i>	<i>Prob{0123579}=4.15127e-005</i>
<i>Prob{5}=8.30254e-005</i>	<i>Prob{0147}=4.15127e-005</i>	<i>Prob{0123789}=8.30254e-005</i>
<i>Prob{6}=0.000207564</i>	<i>Prob{0149}=0.000124538</i>	<i>Prob{0124567}=8.30254e-005</i>
<i>Prob{7}=0.000207564</i>	<i>Prob{0157}=4.15127e-005</i>	<i>Prob{0124568}=4.15127e-005</i>
<i>Prob{8}=0.000207564</i>	<i>Prob{0159}=8.30254e-005</i>	<i>Prob{0124569}=4.15127e-005</i>
<i>Prob{9}=0.000581178</i>	<i>Prob{0179}=8.30254e-005</i>	<i>Prob{0124678}=4.15127e-005</i>
<i>Prob{01}=0.000249076</i>	<i>Prob{0347}=4.15127e-005</i>	<i>Prob{0125679}=4.15127e-005</i>
<i>Prob{03}=4.15127e-005</i>	<i>Prob{01236}=0.000207564</i>	<i>Prob{0134567}=4.15127e-005</i>
<i>Prob{09}=4.15127e-005</i>	<i>Prob{01246}=0.000290589</i>	<i>Prob{0134569}=8.30254e-005</i>
<i>Prob{12}=4.15127e-005</i>	<i>Prob{01248}=4.15127e-005</i>	<i>Prob{0134579}=0.000290589</i>
<i>Prob{13}=0.000124538</i>	<i>Prob{01256}=0.000415127</i>	<i>Prob{0135679}=8.30254e-005</i>
<i>Prob{17}=4.15127e-005</i>	<i>Prob{01259}=4.15127e-005</i>	<i>Prob{01234567}=0.000332102</i>
<i>Prob{18}=4.15127e-005</i>	<i>Prob{01269}=8.30254e-005</i>	<i>Prob{01234568}=0.000124538</i>
<i>Prob{19}=0.000124538</i>	<i>Prob{01345}=0.000166051</i>	<i>Prob{01234569}=0.000747229</i>
<i>Prob{28}=4.15127e-005</i>	<i>Prob{01349}=4.15127e-005</i>	<i>Prob{01234578}=0.000124538</i>
<i>Prob{36}=4.15127e-005</i>	<i>Prob{01379}=4.15127e-005</i>	<i>Prob{01234579}=4.15127e-005</i>
<i>Prob{49}=8.30254e-005</i>	<i>Prob{01478}=4.15127e-005</i>	<i>Prob{01234589}=4.15127e-005</i>
<i>Prob{59}=0.000166051</i>	<i>Prob{01789}=4.15127e-005</i>	<i>Prob{01234678}=0.000332102</i>
<i>Prob{78}=4.15127e-005</i>	<i>Prob{03479}=8.30254e-005</i>	<i>Prob{01234679}=8.30254e-005</i>
<i>Prob{79}=8.30254e-005</i>	<i>Prob{012345}=4.15127e-005</i>	<i>Prob{01234789}=4.15127e-005</i>
<i>Prob{012}=4.15127e-005</i>	<i>Prob{012346}=0.000124538</i>	<i>Prob{01235678}=0.000249076</i>
<i>Prob{013}=0.000166051</i>	<i>Prob{012349}=4.15127e-005</i>	<i>Prob{01236789}=4.15127e-005</i>
<i>Prob{014}=4.15127e-005</i>	<i>Prob{012356}=0.000124538</i>	<i>Prob{01245679}=8.30254e-005</i>
<i>Prob{015}=4.15127e-005</i>	<i>Prob{012379}=8.30254e-005</i>	<i>Prob{01345679}=0.000124538</i>
<i>Prob{017}=0.000166051</i>	<i>Prob{012456}=8.30254e-005</i>	<i>Prob{01345789}=0.000124538</i>
<i>Prob{018}=4.15127e-005</i>	<i>Prob{012478}=4.15127e-005</i>	<i>Prob{012345678}=0.00319648</i>
<i>Prob{019}=0.000290589</i>	<i>Prob{012479}=4.15127e-005</i>	<i>Prob{012345679}=0.00236623</i>
<i>Prob{027}=4.15127e-005</i>	<i>Prob{012579}=4.15127e-005</i>	<i>Prob{012345689}=0.000249076</i>
<i>Prob{078}=8.30254e-005</i>	<i>Prob{012678}=8.30254e-005</i>	<i>Prob{012345789}=0.00045664</i>
<i>Prob{128}=4.15127e-005</i>	<i>Prob{012679}=8.30254e-005</i>	<i>Prob{012346789}=4.15127e-005</i>
<i>Prob{139}=8.30254e-005</i>	<i>Prob{012789}=8.30254e-005</i>	<i>Prob{012356789}=8.30254e-005</i>
<i>Prob{159}=4.15127e-005</i>	<i>Prob{013457}=0.000166051</i>	<i>Prob{013456789}=0.000124538</i>
<i>Prob{179}=4.15127e-005</i>	<i>Prob{013459}=0.000249076</i>	<i>Prob{0123456789}=0.979078</i>
<i>Prob{278}=8.30254e-005</i>	<i>Prob{013479}=0.000124538</i>	<i>Prob{none}=0.00149446</i>
<i>Prob{356}=4.15127e-005</i>	<i>Prob{013679}=4.15127e-005</i>	

Table 9 (cont.)

Meter data		
$Prob\{0\}=0.00132802$	$Prob\{9\}=0.0139442$	$Prob\{45\}=0.00132802$
$Prob\{1\}=0.0139442$	$Prob\{01\}=0.00132802$	$Prob\{015\}=0.000664011$
$Prob\{3\}=0.00796813$	$Prob\{34\}=0.00265604$	$Prob\{019\}=0.000664011$
$Prob\{4\}=0.00332005$	$Prob\{35\}=0.00132802$	$Prob\{145\}=0.000664011$
$Prob\{5\}=0.00398406$	$Prob\{36\}=0.000664011$	$Prob\{0125\}=0.000664011$
		$Prob\{none\}=0.945551$

Now we can compute the error probabilities with the joint distributions using a simple analytical model. Let α denote a subset of signatures and $Pr\{\alpha\}$ denote the joint probability of a packet being identified as malicious by all signatures in α but not recognized by others. Then the probability that an executable packet is incorrectly classified as meter data is:

$$P_{false_negative} = \sum_{\#\alpha < K} Pr\{\alpha\}$$

Similarly, the probability that a meter data is misclassified as executable is:

$$P_{false_positive} = \sum_{\#\alpha \geq K} Pr\{\alpha\}$$

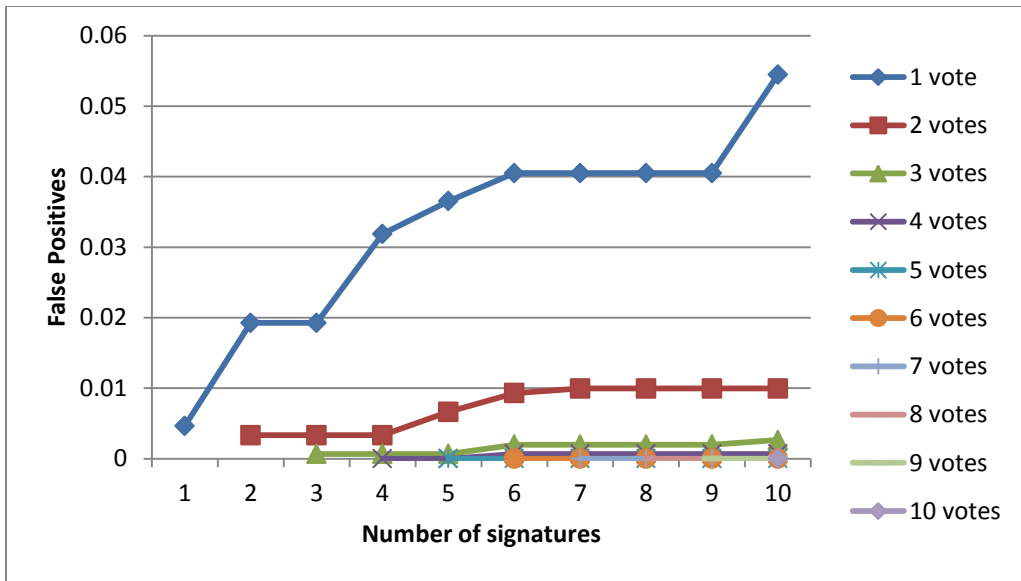


Figure 18 False positives of using multiple signatures.

Based on multi-signature joint distributions, we calculate the error probabilities for false positives and false negatives. Figure 18 shows the false positive probability for each K out of N schemes. The results show that the 1-vote scheme has high error probability in most of the cases. If we allow the policy engine to drop the packet based on one vote, there will be at least 1.9% of the legitimate traffic loss. Apparently the 1-vote scheme is not what we want. More votes are needed to reduce the false positives.

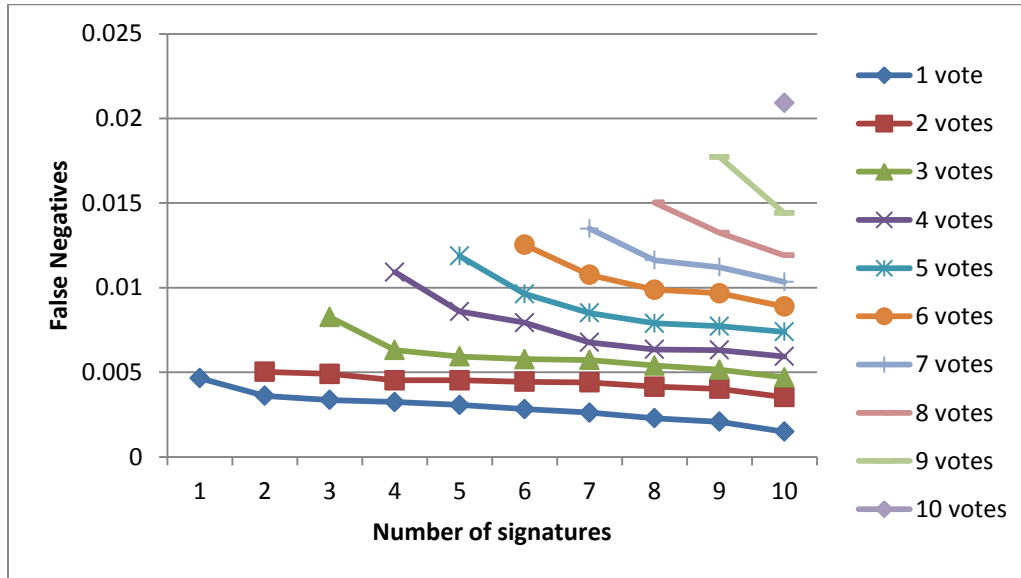


Figure 19 False negatives of using multiple signatures.

From Figure 19 we can see that the more signatures we use the lower error rate we achieve. However, involving too many signatures might slow down the policy engine. Another observation is that if more votes are needed to drop the packets, the error probability becomes higher. These two figures also help us to choose the best strategy for our policy engine. For example, if we want to use 10 signatures, then the 3-vote scheme seems better than others. The 3-vote scheme gives us 0.002656 false positives and 0.004691 false negatives, which provides a good balance of two errors. Note that this scheme also surpasses most of single-signature approaches shown in Table 9 and gives the best error balance.

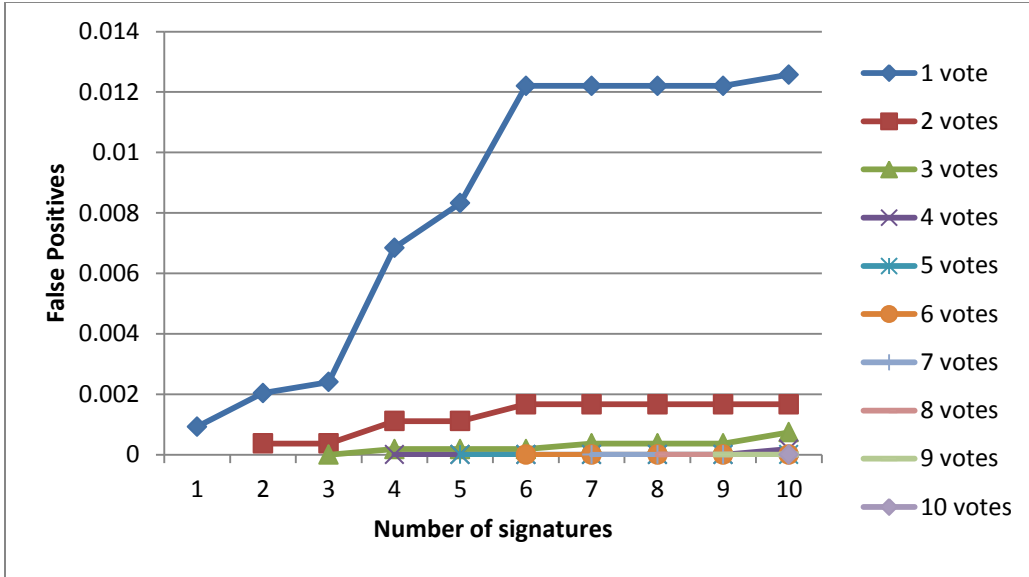


Figure 20 Experimental result of false positives.

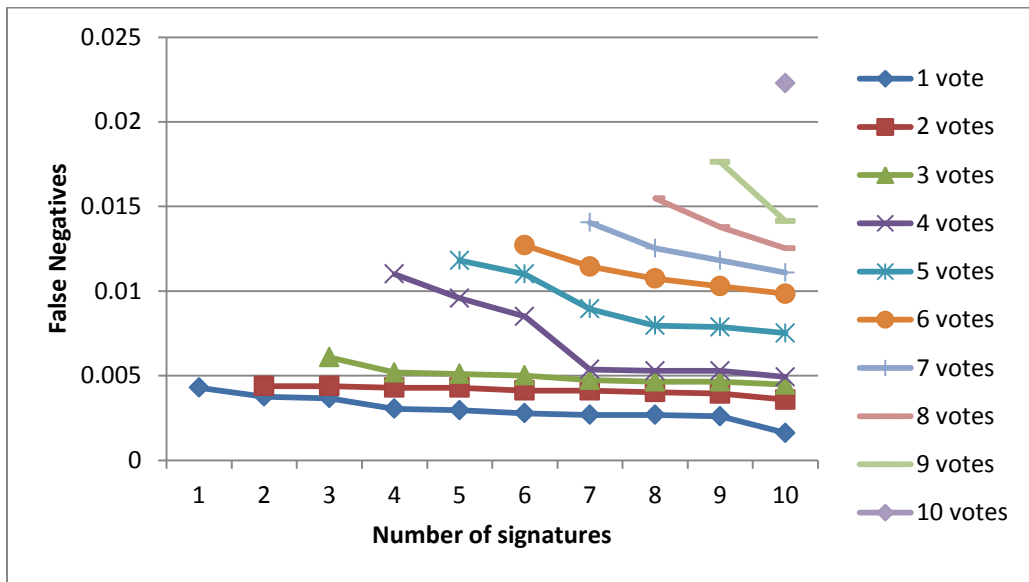


Figure 21 Experimental result of false negatives.

To verify the error probability results, we experiment with the different sets of data. Instead of using joint distributions, the experiments evaluate the multiple signatures errors directly. The errors in Figure 20 and Figure 21 match the results from previous experiments.

9.5.3 Packet Size Sensitivity

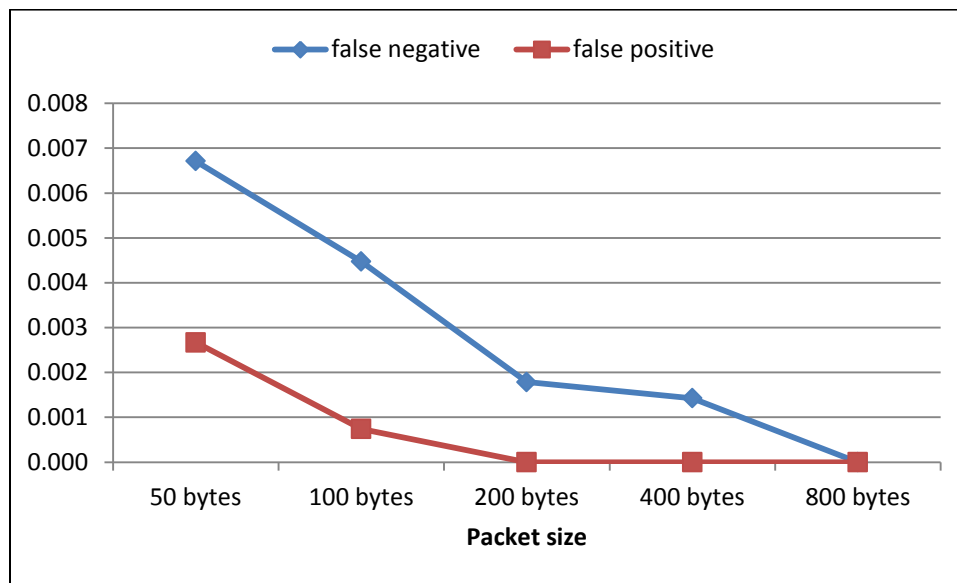


Figure 22 Packet size sensitivity.

In this section, we test how performance changes with various packet sizes. The experiment is conducted with 10 signatures and, three votes are required to drop the packet. Figure 22 shows that large packet size is helpful in reducing both false negatives and false positives because the error rates for a single signature drop when packet size increases.

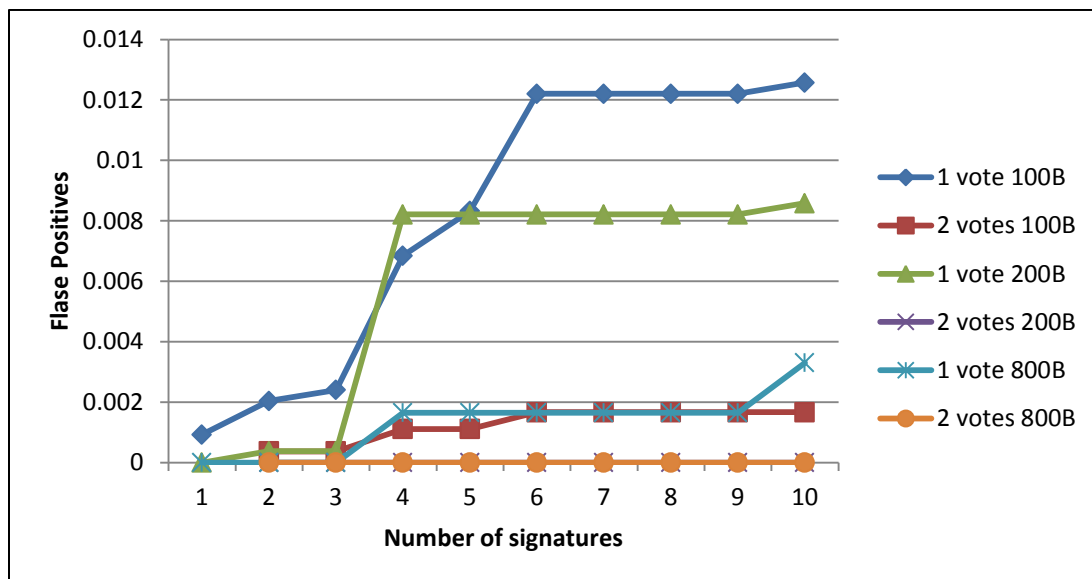


Figure 23 False positives of different schemes with various packet sizes.

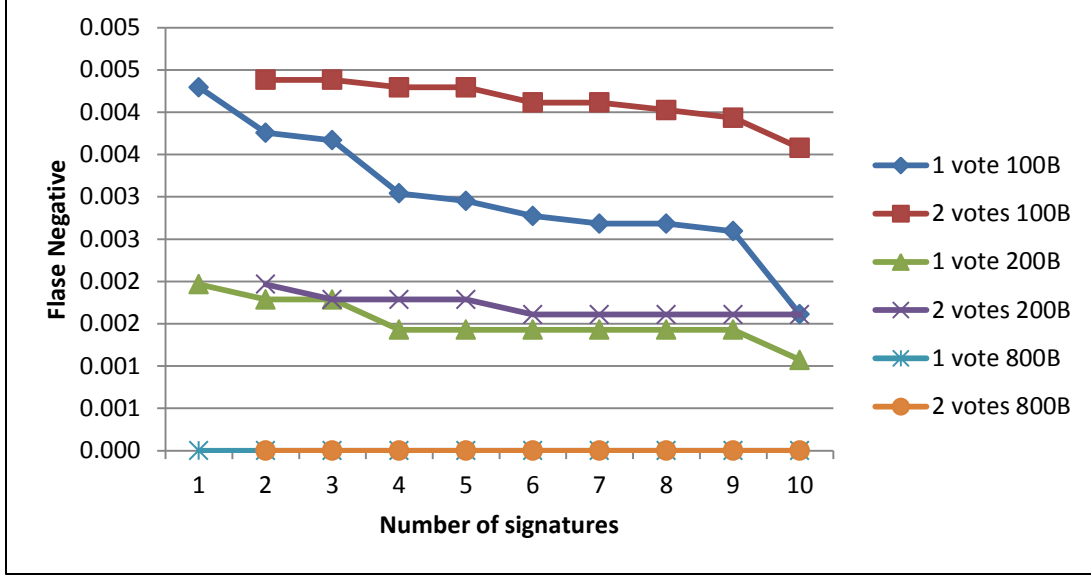


Figure 24 False negatives of different schemes with various packet sizes.

In this section, we present more multi-signature schemes to show their sensitivity to packet size. Due to space limitation, we show 1-vote and 2-vote schemes only. From Figure 23 and Figure 24 we have similar observations on packet size sensitivity.

9.6 Malware File Detection Using Signatures

In previous sections, we have designed and evaluated algorithms to reduce the errors in ARM packet detection. However, the malware file is carried by more than one packet. Even if we cannot successfully identify all of its packets, removal of any packet will inhibit the malware transfer. Then the probability that at least one packet is detected increases with transfer size. In this section, we will analyze the error probability on malware file transfer based on its packet identification error probability.

The malware file identifying process is a Bernoulli trial on sequence of packets with success probability p . Assume that we have a file of size N and it is transmitted in packets of size n . Then the number of packets which are identified correctly by our mechanism is binomial distributed:

$$\text{binomial}(N/n, p)$$

Since we have computed the probability of successfully identifying a malware packet $(1-p')$ using k signatures where p' is the error probability, the probability of detecting and dropping at least c packets of malware (size N , pkt_size n) is

$$\Pr\{binomial(N/n, 1 - p') \geq c\} = \sum_{i=c}^{[N/n]} \binom{[N/n]}{i} (1 - p')^i (p')^{[N/n]-i}$$

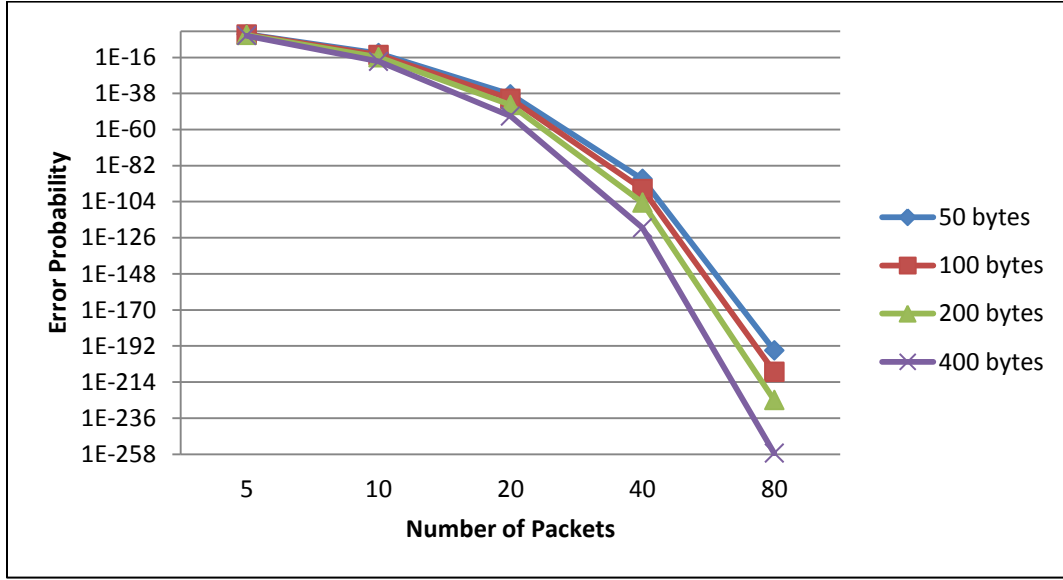


Figure 25 Error probability of identifying an ARM file.

Based on the analytical model, we compute the error probabilities for 10 signatures and the 3-vote scheme. Figure 25 shows the probability of misclassifying an ARM executable file with various packet sizes. In this scenario we consider an ARM file identified correctly if and only if at least five of its packets are detected and dropped. Our method works better with the large amounts of packets and the large packet size.

10. Implementation and Performance

10.1 Implementation

We implemented a prototype of the proposed policy engine based on an open source implementation of DLMS/COSEM provided by GuruX [19]. The policy engine is implemented between a DLMS/COSEM library and applications using the GuruX library. The policy engine defines the same API used by the applications, and each function in the API is used as a wrapper function to intercept all the traffic. The API functions will call the function matched in the DLMS/COSEM library after performing necessary policy checks.

To provide transparency to DLMS/COSEM application developers, we use the Ant tool [20] to build our Java application using source-code-to-source-code translation, which automatically changes DLMS/COSEM library calls to methods in the policy engine. The source-to-source translation can be achieved by Ant's built-in filter function so that the developers do not need to remember the symbols in the policy engine library. We simply added the following XML code to the pre-compile section in Ant's build file to translate the source code before compilation.

```
<copy todir="build/target-src" filtering="true" overwrite="true">
  <fileset dir="src" />
  <filterset begintoken="c" endtoken="(">
    <filtersfile file="myfilters"/>
  </filterset>
</copy>
```

The PE.filters file defines token-value pairs that map GuruX library symbols to the policy engine Symbols. Figure 26 shows the overview of DLMS/COSEM system with our policy engine.

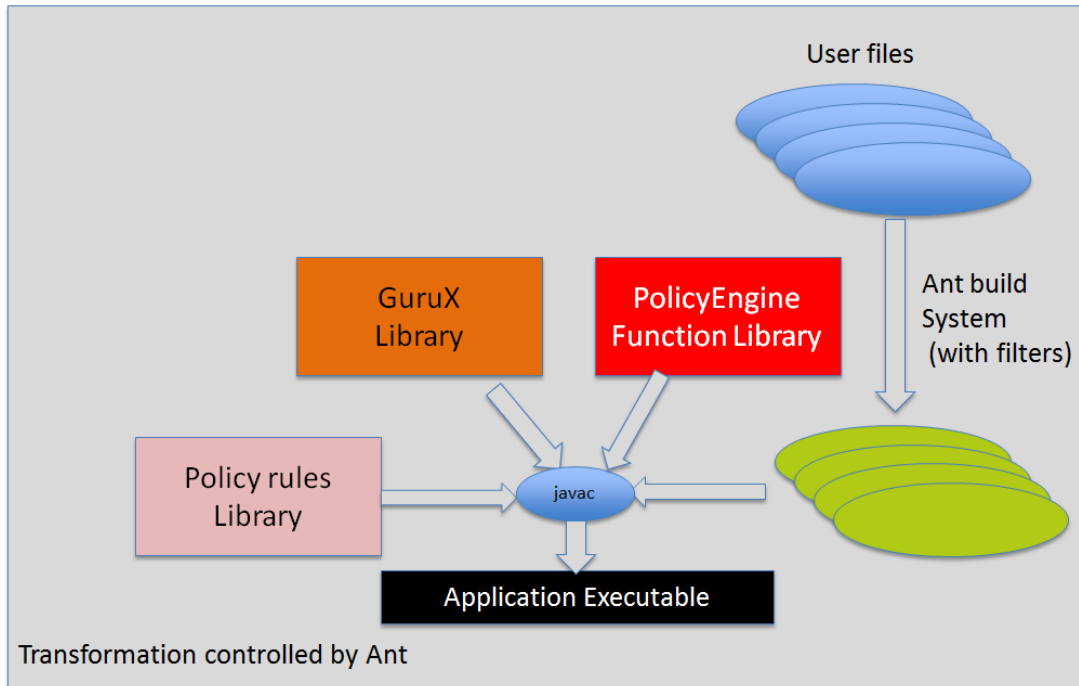


Figure 26 Source code translation using Ant filters.

Table 10 shows the mappings that are currently used in our policy engine prototype. It is used as a properties file for the Ant filter to load tokens from.

Table 10 Mapping GuruX calls to policy engine calls

GuruX calls	Policy engine calls
client.getDataFromPacket	pedlms.getDataFromPacket
client.receiverReady	pedlms.receiverReady
client.getGXDLMSClientObject	pedlms.getGXDLMSClientObject
client.getDataFromPacket	pedlms.getDataFromPacket
client.getGXDLMSClientObject	pedlms.getGXDLMSClientObject
client.SNRMRequest	pedlms.SNRMRequest
client.AARQRequest	pedlms.AARQRequest
client.parseAAREResponse	pedlms.parseAAREResponse
client.parseUAREsponse	pedlms.parseUAREsponse
client.getGXDLMSClientObject	pedlms.getGXDLMSClientObject
client.read	pedlms.read
client.getValue	pedlms.getValue
client.getGXDLMSClientObject	pedlms.getGXDLMSClientObject
client.setObisCodes	pedlms.setObisCodes
client.getObjects	pedlms.getObjects
client.parseObjects	pedlms.parseObjects

10.2 Performance Evaluation

We evaluated the overhead for our policy engine to check a set of policy rules and analyze the traffic. We measured the aggregated time that elapsed in performing the policy check on the client side. The experiment included examination of both request and response packets to and from the server in a given session. The session we evaluated included connecting application association, reading implemented object lists of a server, and reading CLOCK/DATA/REGISTER/PROFILE GENERIC objects. We used a machine with eight processors and 16 GB of memory. We repeated the experiment 100 times and got the average processing time.

We have measured the application delay, which is defined as the time period between its object request and response. Without the policy engine, the application delay is 17,308,095 ns on average. After we integrate our policy engine, the application delay increased to 19,464,303 ns on average, which is an 12.458% increase compared to the non-PE scenario. We have also evaluated the pure processing time consumed by the GuruX library and policy engine library, respectively. Since the processing time does not include communication delay, the processing time of the policy engine is much higher than the GuruX due to extensive traffic analysis. Our experiments show that the average processing time of using the policy engine is 2,441,325 ns which is about 10 times of that the GuruX's 242,874 ns processing time. As previously mentioned, we set up both client and server on the same physical machine to minimize communication delay. In the real AMI system, the communication time is much longer than the processing time. Therefore, the fraction of policy engine processing time is smaller than what we saw in the experiments.

In addition to the previous performance experiments, we measured the performance overhead in an environment with network delay. In this test, the DLMS/COSEM client and server are installed on two physical machines within the Illinois.edu domain. Our experiments show that, due to network latency, the application delay caused by the policy engine drops to 9.693% and the processing time becomes 8.97 times of the original GuruX processing time.

The policy engine checks the fixed set of rules and does a relatively predictable amount of analysis work. And it is obvious that the process of checking one chunk of data is independent of others. Thus, we expect the performance overhead is proportional to the size of data being processed. We increased the data size up to 10 times of the default size by appending more items to the DLMS/COSEM data object. We observed that the application overhead is slightly increased but still around 10%. We believe that

the overhead caused by the policy engine is not very sensitive to data size within the range of DLMS/COSEM's normal usage.

11. Conclusion

This work proposed a new policy engine that would not allow any unwanted traffic, especially malware, either in or out. The policy engine prototype was designed according to the standard DLMS/COSEM protocol and the statistical features of malware. Fortunately, the policy engine can be easily extended to similar protocols and systems such as wireless network AMI system using C12.22. The standard protocol analysis monitors abnormal behavior in the systems. Based on the obvious traits (i.e., packing and execution) of malware, the statistical analysis aims to detect malicious code in metering data. Our policy engine can detect or block any malicious information or code with negligible performance penalty. Our research shows that metering data and malware demonstrate different statistical characteristics in entropy and byte patterns. With a version of open-source DLMS/COSEM from GuruX, we manage to incorporate our policy engine to test its effectiveness and efficiency. We also suggest a method of source-to-source translation to help the application developer to work with the policy engine easily. The experiments show that even with a less powerful desktop machine, our policy engine only involves minor overhead, which is about 12.458% more time used with respect to the GuruX application without the policy engine.

References

- [1] M. LaMonica. Obama signs stimulus plan, touts clean energy. http://news.cnet.com/8301-11128_3-10165605-54.html
- [2] Smart grid dark side. <http://greeneconomypost.com/smart-grid-dark-side-1249.htm>
- [3] Who Controls the Off-Switch, 2010. <http://www.senseofsecurity.com.au/presentations/>
- [4] ANSI c12.22, American National Standard Protocol Specification for interfacing to data communication networks. <http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+C12.22-2008>
- [5] The official IEC standard. <http://webstore.iec.ch/webstore/webstore.nsf/artnum/031526>
- [6] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Proceedings of the 7th Conference on USENIX Security Symposium*, 1998.
- [7] M. Davis, "Smartgrid device security: Adventures in a new medium," in *Proceedings of the Black Hat Technical Security Conference*, 2009.
- [8] 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 (ACSAC '10)*, 2010.
- [9] F. Cleveland, "Cyber security issues for Advanced Metering Infrastructure(AMI)," in *Proceedings of IEEE conference of Power and Energy Society General Meeting*, 2008.
- [10] R. H. Mustafa, M. Xu, W. Xu, R. Miller, and M. Gruteser, "Neighborhood watch: Security and privacy analysis of automatic meter reading systems," in *Proceedings of the ACM conference on Computer and communications security (CCS)*, Raleigh, NC USA, 2012.
- [11] W. Wang and Z. Lu, "Cyber security in the smart grid: Survey and challenges," *Computer Networks*, pp. 415–427, 2013.
- [12] W. Yang, N. Li, and Q. Yuan, "Minimizing private data disclosures in the smart grid," in *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, 2012.
- [13] X. Li, X. Liang, R. Lu, X. Shen, X. Lin, and H. Zhu, "Securing smart grid: Cyber attacks, countermeasures, and challenges," *Communications Magazine, IEEE*, vol. 50, no. 8, pp. 38–45, 2012.
- [14] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

- [15] Kosut, L. Jia, R. Thomas, and L. Tong, "Malicious data attacks on smart grid state estimation: Attack strategies and countermeasures," in *IEEE International Conference on Smart Grid Communications*, 2010.
- [16] W.-J. Li, K. Wang, S. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," in *Proceedings of the IEEE Workshop on Information Assurance and Security*, 2005.
- [17] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 40–45, Mar. 2007.
- [18] TCIPG. <http://tcipg.org>
- [19] GuruX. <http://www.gurux.fi>
- [20] Apache Ant. ant.apache.org