Design of Policy Engine to Prevent Malware Propagation in Advanced Metering Infrastructure

Younghhee Park
Computer Engineering Department
San Jose State University
Introduction
Introduction

• Advanced Metering Infrastructure (AMI)

Smart Meters → Data Concentrator Unit (DCU) → Headend

WiFi, PLC, Zigbee or Radio

NAN (Neighborhood Area Network) → Utility Data Center
- Demand Response Center
- Customers Information Systems
- Meter Data Management

WAN (Wide Area Network)
Introduction

• Goals of Advanced Metering Infrastructure (AMI)
  – Reliability
    • Smart metering
    • Automated real-time monitoring
  – Efficiency
    • Demand response and control
    • Active energy management
    • Remote connect/disconnect
  – Security
    • Improved by monitoring and reliability
Cyber Attacks in AMI

• **Network availability attack**
  – Overwhelm the communication and computational resources

• **Data attacks**
  – Insert, alter or delete data in the network traffic

• **Privacy attacks**
  – Learn users’ private information by analyzing electricity usage data

• **Device attacks**
  – Compromised or control a device
Cyber Attacks in AMI

• Malware (Malicious Software)
  – Stuxnet, Aurora worm in industrial systems
  – Warning from Blackhat, McAfee
  – Control systems and disrupt operations
  – Launch malicious commands
    • Blackout, remote disconnects
  – Infect other systems in AMI
Policy Engine

• What is Policy Engine?
  – A set of policy rules
  – Detection/Prevention of Malware carried by an application layer protocol
    • Malware propagation during communications
  – Monitor bi-directional communications
    • Client request (DCU, Headend)
    • Server response (Smart meters)
Policy Engine

DCU
Client

Application Layer
Supporting Communication Profiles

Application 1  Application 2  ...  Application N

Request

Network

DLMS/COSEM protocol
TCP/UDP, PLC, HDLC

Smart Meter
Server

Application Layer
Supporting Communication Profiles

Logical Device 1  Logical Device 2  ...  Logical Device N

Response
DLMS/COSEM Protocol

• An standard application protocol for AMI
• **DLMS** (Device Language Message Specification) – define **objects** found in meters
• **COSEM** (Companion Specification for Energy Metering) – Define **communication** between meters and devices needing meter information
DLMS/COSEM Protocol

- **Interface classes** (Class ID)
  - All meter data as objects
  - Object Identification system (OBIS)
    - OBIS code is called a logical name (1st attribute value)
    - [EX] Reading energy value
      - Register (class_id : 3)
      - Obis code is “1.1.1.8.0.255”
      - Reading second attribute value
    - 70 standard interface classes
Motivation for Policy Engine

• Compromised applications or devices
• Propagate malware by hiding itself into DLMS/COSEM traffic
  – Violation of Packet formats
  • Specification of DLMS/COSEM application layer
  – No metering data
  – Containing malware features
System Architecture of Policy Engine

Data Collection

- Interface Classes & OBIS codes
- Control Command Types
- System Configurations

Policy Rule Monitor

- Policy Rules
- Standard Protocol Analyzer
- Statistical Analyzer

Profiling

Monitoring

Logging

Metering Data?
System Architecture of Policy Engine

- **Data collection system**
  - OBIS, Command types, system configurations

- **Policy Rule Monitoring system**
  - Policy rules from the standard protocol
  - Policy rules from two statistical features

- **Logging system**
  - Recording malicious data
Standard Protocol Analysis

• DLMS/COSEM protocol
  – It creates application data
    • APDU (application protocol data unit)
  – Policy rules
    • Syntactic policy rules
    • Semantic policy rules
    • Access policy rules
    • Communication policy rules
Standard Protocol Analysis

- DLMS/COSEM Protocol
  - Syntactic/Semantic/Access/Communications
- Malware overwrite the APDU
Statistical Analysis

• Two features of malware
  – Packed or encrypted binaries are around 90%
  – Executable format

• Smart meters
  – Low-computational power
  – Limited bandwidth and resources

• Statistical analysis tools
  – Entropy analysis
  – N-gram analysis
Statistical Analysis

• Entropy analysis
  – Bintropy
    • Shannon entropy
    • Detection method for packing and encrypted binaries
    • Packed and encrypted binaries have greater than 6.67 entropy
  – Metering data have low entropy less than 6.
    • Evaluating the frequency of each byte in fixed block size
Statistical Analysis

• N-gram analysis
  – FilePrint
    • Each file type has its own distinct 1-gram distribution
  – 1-gram distribution of metering data is different from the one of malware
  – Measuring dissimilarity(distance) between two distributions
    • Manhattan distance and Mahalanobis distance
Statistical Analysis

• N-gram analysis
  – **Trained Model** for malware
    • K-mean clustering & Manhattan distance
    • EX) 10 malware files, K=2 (2 Trained models)
      – Pick 2 random files -> evaluate distributions -> update distribution by K-mean clustering and manhattan distance

  – **Testing** Model for metering data
    • Mahalanobis distance to compare to the trained model
    • Distance(testing file, K-model) > threshold
Experiments

• Data samples
  – Trained model for 200 malware samples
  – Real 100 metering data files from the TCIPG testbed

• Policy engine implementation
  – Open source GuruX for DLMS/COSEM
  – Intercept API calls of GuruX

• Experiments
  – Entropy & 1-gram distribution & false positive rate & overhead
Entropy for Metering Data

Entropy of Encrypted Binaries (Bintropy)

Entropy of Packed Binaries (Bintropy)

Paylaod Size (Bytes)

Entropy
1-gram Distribution

One malware sample

One metering data file
False Positive Rate

- Metering data are recognized as malware
- O.17% performance overhead

<table>
<thead>
<tr>
<th>Truncation</th>
<th>50 Bytes</th>
<th>100 Bytes</th>
<th>200 Bytes</th>
<th>500 Bytes</th>
<th>1 KB</th>
<th>2KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=1</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
</tr>
<tr>
<td>K=2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>K=5</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>k=10</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>k=20</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td>1%</td>
<td>2%</td>
</tr>
</tbody>
</table>
Conclusion

• Policy Engine
  – Monitor DMLS/COSEM traffic to detect malware
  – Malware must try to hide as data, but we can ask many policy rules that reveal its presence
  – Use the standard protocol specification and the statistical features of metering data
Current/Future Work

• Improve Accuracy of Malware Detection
  – ARM Instruction Distribution
  – "e" code testing accepted for SmartGridComm 2013

• Performance evaluation on embedded systems
  – Beagle board, Raspberry PI, etc.
Q & A

Thank You!
Interface Class

**Register** class_id=3
- logical_name: octet-string
- value: instance specific...
- reset

**Object**
- **Total Positive Active Energy: Register**
  - logical_name = [1 1 1 8 0 255]
  - value = 1483

- **Total Positive Reactive Energy: Register**
  - logical_name = [1 1 3 8 0 255]
  - value = 57

**Attributes**

**Instantiation**

<table>
<thead>
<tr>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
</tr>
<tr>
<td>Attribute(s)</td>
</tr>
<tr>
<td>1. logical_name</td>
</tr>
<tr>
<td>2. value</td>
</tr>
<tr>
<td>3. scaler_unit</td>
</tr>
</tbody>
</table>

**Specific methods (if required)**

1. reset (data)
Example of Communication

Example 1
- Server ID(3), Client ID(33), GET, Class_ID(1), OBIS(0.0.42.0.255), INDEX(2)
- Server ID(3), Client ID(33), GET, Class_ID(1), Data(Device Name)

Example 2
- Server ID(3), Client ID(33), GET, Class_ID(8), OBIS(0.0.1.0.255), INDEX(2)
- Server ID(3), Client ID(33), GET, Class_ID(1), Data(Current Time)

Example 3
- Server ID(3), Client ID(33), GET, Class_ID(7), OBIS(1.0.99.1.0.255), INDEX(3)
- Server ID(3), Client ID(33), GET, Class_ID(1), Data(Class_ID(8) OBIS(0.0.1.0.0.255), Index, Data, Class_ID(3), OBIS(1.1.21.250.255), Index, Data)
# Result of Bintropy

## Table 1. Computed statistical measures based on four training sets.

<table>
<thead>
<tr>
<th>DATA SETS</th>
<th>AVERAGE ENTROPY</th>
<th>99.99% CONFIDENCE INTERVALS (LOW TO HIGH)</th>
<th>HIGHEST ENTROPY (AVERAGE)</th>
<th>99.99% CONFIDENCE INTERVALS (LOW TO HIGH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted executables</td>
<td>7.175</td>
<td>7.174 – 7.177</td>
<td>7.303</td>
<td>7.295 – 7.312</td>
</tr>
</tbody>
</table>

\[
H(x) = - \sum_{i=1}^{n} p(i) \log_{2} p(i),
\]
FilePrint

• 1-gram distributions
  – 256 distinct bytes from 00000000 to 11111111

• Mahalanobis distance

  In to represent a class of file types. The resul
  eriments indicate that indeed 1-grams per

• Manhattan distance

\[ D(A,B) = \sum_{i=0}^{255} | A_i - B_i | \]
Modeling

- The K-means algorithm that computes multiple centroids is briefly described as follows:

  1) Randomly pick K files from the training data set. These K files (their byte value frequency distribution) are the initial seeds for the first K centroids representing a cluster.

  2) For each remaining file in the training set, compute the Manhattan Distance against the K selected centroids, and assign that file to the closest seed centroid.

  3) Update the centroid byte value distribution with the distribution of the assigned file.

  4) Repeat step 2 and 3 for all remaining files, until the centroids stabilize without any further substantive change.