Operation-Level Traffic Analyzer Framework for Smart Grid

Wenyu Ren
Department of Computer Science, University of Illinois Urbana-Champaign
201 North Goodwin Avenue
Urbana, Illinois 61801-2302
wren3@illinois.edu

Klara Nahrstedt
Department of Computer Science, University of Illinois Urbana-Champaign
201 North Goodwin Avenue
Urbana, Illinois 61801-2302
klara@illinois.edu

Tim Yardley
Information Trust Institute, University of Illinois Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801-2307
yardley@illinois.edu

ABSTRACT

The Smart Grid control systems need to be protected from internal attacks within the perimeter. In Smart Grid, the Intelligent Electronic Devices (IEDs) are resource-constrained devices that do not have the ability to provide security analysis and protection by themselves. And the commonly used industrial control system protocols offer little security guarantee. To guarantee security inside the system, analysis and inspection of both internal network traffic and device status need to be placed close to IEDs to provide timely information to power grid operators. For that, we have designed a unique, extensible and efficient operation-level traffic analyzer framework. The timing evaluation of the analyzer overhead confirms efficiency under Smart Grid operational traffic.

CCS Concepts

•**Security and privacy** → **Network security;** •**Networks** → *Network monitoring;*

Keywords

Smart Grid; network security; traffic analysis

1. INTRODUCTION

For the Smart Grid control networks nowadays, data within networks are usually not visible to the operators and not secured at the same security levels as the communication with external entities. To provide end-to-end security inside the network system, both the end hosts and the network need to be secured. In general purpose network such as the Internet, the end hosts usually have their own security analysis and protection mechanism. Therefore, the analyzers designed for general purpose network [1, 2, 3, 4, 5, 6] only need to provide network analysis capability, i.e., flow/packet or application level traffic analysis. In Smart Grid network, shown in Figure 1, the end hosts are control centers and Intelligent Electronic Devices (IEDs). Since the IEDs are resource-constrained devices, the control centers are usually responsible for both the device status analysis and network traffic analysis. However, waiting for all measurement data to be gathered in one location before processing data for diverse anomaly detection introduces huge delay, which may result in huge losses during security breaches.

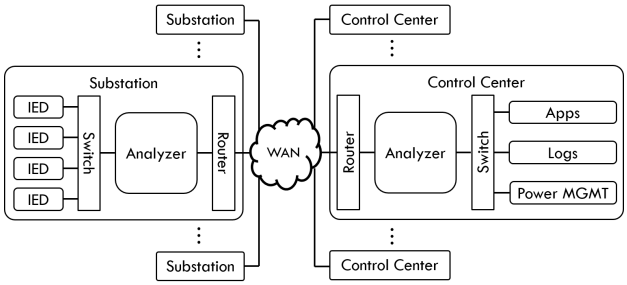


Figure 1: Smart Grid network architecture

Our approach to the problem in Smart Grid networks is the design of an extensible and efficient operation-level traffic analyzer that will execute both flow-level network traffic analysis and operation-level device status analysis to identify anomalies. For network traffic analysis in the flow level, the analyzer is able to track which two hosts are communicating and what application protocol they are using. For device status analysis in the operation level, the analyzer is able to track the operations (e.g., read or write) that are taking place in the industrial control systems protocols like DNP3 and Modbus, and the targets of those operations(e.g., which coil or register value is read/written). The analyzer collects, aggregates and stores these meta data statistics in efficient data structure. Then, it inspects those aggregated data to performs anomaly detection.

We deploy our analyzer at the boundary of the WAN at both control center and substation ends, as it is shown in Figure 1. By providing both network and device analysis ability close to IEDs, we are able to give the power operator more up-to-date view of the whole system and warn them of potential breaches more promptly. To our knowledge, there is no existing analyzer that provides operation-level device analysis. And operation-level device analysis is crucial to Smart Grid networks, since compromised devices can report faked measurement data and execute malicious operations (commands, instructions) which could cause huge damage to the entire system.

2. ANALYZER DESIGN

HotSoS '16 April 19-21, 2016, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-4277-3/16/04.
DOI: http://dx.doi.org/10.1145/2898375.2898396

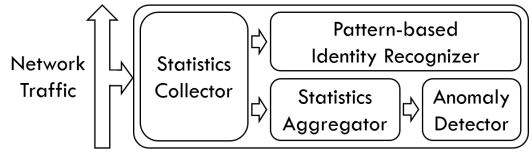


Figure 2: Modular Structure of the Analyzer

While designing the analyzer, there are two challenges crucial to our analyzer: (1)*Extensibility* towards new devices and new protocols using modularization as the Smart Grid networks evolve, (2)*Efficiency* regarding real-time traffic collection and analysis using multiple levels of statistics. The modular structure of the analyzer is shown in Figure 2. The analyzer consists of 4 modules: (1)*Statistics Collector*, (2)*Statistics Aggregator*, (3)*Anomaly Detector*, (4)*Pattern-based Identity Recognizer*.

The ***Statistics Collector*** examines the network packets and collects 5 levels of statistics: (1)Sender of the packet, (2)Receiver of the packet, (3)Protocol that the packet uses for industrial control, (4)Function (e.g. read or write) that the packet executes in its protocol, (5)Target of the function (e.g. which coil or register). Level 1-3 are flow-level statistics that is used for network traffic analysis, while level 4&5 are operation-level statistics that is used for device status analysis.

Each packet header will go through four metadata extractors in order. The four extractors will extract the statistics of level 1&2, 3, 4 and 5, respectively. The first extractor is a general one which extracts *sender* and *receiver* information. The other three, on the other hand, are protocol and device specific and are responsible for extracting *protocol*, *function* and *target* information, respectively. Currently, we have extractors for two industrial control protocols, DNP3 and Modbus. An *item_gen* event will be triggered after the packet is processed by the last extractor, which contains all the statistics extracted by the current and all former extractors.

If new operations in new protocols need to be supported, nothing in the analyzer but the last three extractors need to be updated, which provides significant extensibility. Moreover, users can easily scale the number of levels of statistics to collect. If the user is only interested in the flow-level information, the analyzer can be configured into a general network analyzer by only collecting the upper three levels of statistics. This can largely speed up the collector and make the analyzer more efficient, since the packet needs to go through the first two extractors only instead of all four in this case.

The ***Statistics Aggregator*** aggregates the information of each packet and constructs a tree structure $Tree_{new}$ to store the aggregated statistics. Each tree structure $Tree_{new}$ corresponds to statistics aggregated over certain period of time T_p and each node corresponds to statistics of a specific kind of packets. An example of the data structure is shown in Figure 3. Each node (leaf and internal) in the tree includes the following fields: (1)Info string IS, (2)Accumulated info string AIS, (3)Packet count PC, (4)Byte count BC, (5)Response ratio RR (Func node only), (6)Response delay RD (Func node only). IS is the value of the corresponding statistics level. For example, IS of *sender* level is its IP address and IS of *function* level is the function name. AIS of a node, on the other hand, is constructed by ISes of nodes on the path from the root to itself. And the node stores statistical data of the kind of packet represented by its AIS. For example, the node labelled "G_1" in Figure 3 has AIS of "$S_1 - R_1 - P_1 - F_1 - G_1$" and therefore stands for packets that sender S_1 sends to receiver R_1 using protocol P_1 with function F_1 performed on target G_1. The other four fields are data fields used to store statistic data of that kind of packets corresponding to the node during that period T_p. PC is the total number of the packets, while BC is the total amount of bytes of the packets. RR and RD only exist for *function* level nodes and are the ratio of responded request functions and the delay of the response, respectively.

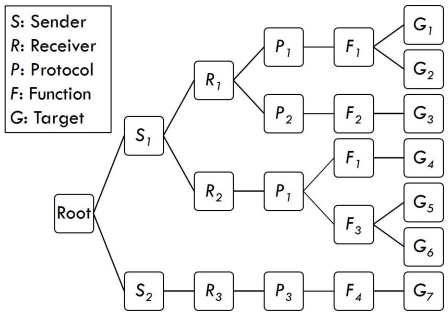


Figure 3: Statistics structure

In the workflow of the Statistics Aggregator, the *item_gen* event from the Statistics Collector is fed into an item counter, which gets the information about the packet in the event and increases the corresponding nodes' counters. There is also an aggregator which runs every T_p, aggregating the results during that period as well as constructing the statistics structure. After the aggregator finishes the aggregation and construction, it triggers an *aggre_finish* event which includes the tree structure $Tree_{new}$ of that period.

The ***Anomaly Detector*** triggers alarms when anomalous traffic is seen in the network. Our current anomaly detector utilizes a threshold-based algorithm, named *Normal Tree*. The core idea of the algorithm is constructing a 'normal' tree $Tree_k$ which represents the normal traffic and using it as a baseline. The next tree $Tree_{new}$, constructed by the Statistic Aggregator, is then compared to the baseline to detect any potential anomaly.

The algorithm has two phases: initialization and anomaly detection. In the initialization phase, which is the first k T_p periods, the algorithm just merges the k trees and constructs the normal tree $Tree_k$. The structure of the normal tree $Tree_k$ is similar to $Tree_{new}$ except that we store a mean value μ and a standard deviation value σ for each statistic field (PC, BC, RR, RD) in each node N. In the anomaly detection phase, we compare $Tree_{new}$ with $Tree_k$. One node N in $Tree_{new}$ is considered to be the "same" with another in $Tree_k$ if they have the same AIS. To compare the two same nodes in two trees, we assign anomaly scores to each data field of them. Suppose the field in $Tree_{new}$ has a value of X and the corresponding field in $Tree_k$ has value μ and σ. Utilizing the Chebyshev's inequality, we define anomaly score AS as follows:

$$AS(X,\mu,\sigma) = \begin{cases} 1 - \frac{\sigma^2}{|X-\mu|^2} & \text{if } |X-\mu| > \sigma \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The anomaly score is in the range $[0, 1]$ and a higher score represents more abnormal behavior. The algorithm then compares that score with a predefined threshold θ and decide whether to trigger an alarm or not. The anomaly detection phase of the algorithm is shown in Algorithm 1.

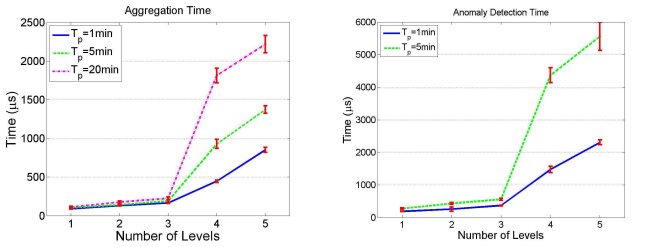
Algorithm 1 Normal Tree Algorithm

procedure AnomalyDetect($Tree_{new}, Tree_k, \theta$)
  Traverse $Tree_{new}$ and $Tree_k$ simultaneously in pre-order.
  for each node N: **do**
    if N exists in both $Tree_{new}$ and $Tree_k$ **then**
      Use Equation 1 to calculate AS for each data field of N and compare them with θ
    else if N exists in $Tree_{new}$ but not in $Tree_k$ **then**
      Assign $AS = 1$ to N instead of each data field and compare it with θ
    else if N exists in $Tree_k$ but not in $Tree_{new}$ **then**
      Create a dummy node N in $Tree_{new}$ with all data fields set to zero. Then use Equation 1 to calculate AS for each data field and compare them with θ
    end if
  end for
end procedure

The ***Pattern-based Identity Recognizer*** identifies the type of the traffic source and destination by monitoring certain request and response patterns in the traffic statistics. It consists of a request-response coupler and a recognition rule matcher. The request-response coupler analyzes the packet statistics in the *item_gen* event. It couples each pair of request and response and constructs a variable which consists of both of their functions. For each pair of request and response, this variable is checked by the recognition rule matcher to see whether it matches the function pairs given by the recognition rules or not. If a match is found, the identities of the requester and the responder are also given by the corresponding matched rule and output by the matcher.

3. TIME OVERHEAD EVALUATION

We use a network analysis framework, called Bro, to implement our analyzer. All experiments run on a synthetic Modbus trace set. For the Statistics Collector, we measure the processing time of each individual packet and define it to be the total runtime of the collector. Results show that even if we collect statistics from all 5 levels, the runtime is still short enough for the packets to be processed in communication line speed. Moreover, reducing the levels has a significant effect on the decrease of total runtime. For the Traffic Statistics Counter, we are interested in the aggregation time, which is the time for the aggregator to aggregate the counters and construct the current tree structure. The measured time with different levels and aggregation periods are shown in Figure 4a. It is clearly that the aggregation time increase as the number of levels increases. And a larger T_p results in a longer aggregation time. For the Anomaly Detector, the time to run the Normal Tree algorithm for one period is denoted by the anomaly detection time. Varying the number of levels and aggregation period T_p, we have different anomaly detection time shown in Figure 4b. Similarly, the anomaly detection time increase as the number of levels increases or T_p increases.



(a) Aggregation time (b) Anomaly detection time

Figure 4: Time overhead with different levels and aggregation period

As it is shown in the above results, the effect of number of levels is significant on the time overhead of different modules. Therefore, always using the least necessary number of levels can save non-negligible amount of time and produces the highest processing speed. In this way, efficiency can be achieved by the analyzer.

4. CONCLUSION

In this paper, we show that our extensible and efficient operation-level traffic analyzer framework inside of Smart Grid networks provides network traffic and device status analysis.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000676.

6. REFERENCES

[1] W. Erhard, M. Gutzmann, and H. Libati. Network traffic analysis and security monitoring with unimon. In *High Performance Switching and Routing, 2000. ATM 2000. Proceedings of the IEEE Conference on*, pages 439–446, 2000.

[2] R. Handayanto, Haryono, and J. Prianggono. Real-time neural network-based network analyzer for hotspot area. In *Advanced Computer Science and Information System (ICACSIS), 2011 International Conference on*, pages 323–330, Dec 2011.

[3] D. Keim, F. Mansmann, J. Schneidewind, and T. Schreck. Monitoring network traffic with radial traffic analyzer. In *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 123–128, Oct 2006.

[4] T. McGregor, H.-W. Braun, and J. Brown. The nlanr network analysis infrastructure. *Communications Magazine, IEEE*, 38(5):122–128, May 2000.

[5] M. Rahman, Z. Khalib, and R. Ahmad. A portable network traffic analyzer. In *Electronic Design, 2008. ICED 2008. International Conference on*, pages 1–6, Dec 2008.

[6] E. Sharafuddin, N. Jiang, Y. Jin, and Z.-L. Zhang. Hospital: Host and network system profiler and internet traffic analyzer. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 420–424, Dec 2010.