An IEC 61850 MMS Traffic Parser for Customizable and Efficient Intrusion Detection

Heng Chuan Tan*,Vyshnavi Mohanraj*,Binbin Chen†*,Daisuke Mashima*,Shing Kham Shing Nan†,Aobo Yang†
*Advanced Digital Sciences Center,Singapore †Singapore University of Technology and Design,Singapore
Email: {hc.tan,vyshnavi.m,daisuke.m}@adsc-create.edu.sg, binbin_chen@sutd.edu.sg,
{aobo_yang,shing_nan}@mymail.sutd.edu.sg

***Abstract*—Manufacturing Message Specification (MMS) protocol is widely used in IEC 61850-based substations to improve process automation. However, it could be vulnerable to various cyber threats. A common defense solution is to deploy intrusion detection systems (IDSes) to analyze network traffic for anomalies. However, several challenges remain for designing a protocol parser for IDS to dissect MMS packets, such as the need to support many MMS services and the complex data structure. Moreover, processing every MMS packet may overwhelm the IDS to impact the throughput and latency. In this work, we develop an MMS parser for the open-source Zeek IDS to analyze MMS traffic and detect intrusions. We explain the challenges of parsing MMS packets and detail our design choices. To reduce the processing load, we implement filtering rules in our parser to customize which MMS packets are used by Zeek rules for intrusion analysis. We formulated test cases to validate our parser's correctness and conducted experiments to evaluate its throughput and latency. Our results show that custom filtering of MMS packets can achieve higher throughput and lower delay compared to no filtering. We provide a case study to demonstrate how the parsed data can be used for designing IDS rules.**

I. Introduction

The electrical substation has undergone several design changes in recent years [1]. One key advancement is the adoption of IEC 61850 standard [2], which aims to replace all hardwired connections with more advanced Ethernet and TCP/IP technologies to improve communications. The main contribution of IEC 61850 is the definition of abstract data models that allow devices from different vendors to interoperate. The current protocol mappings are manufacturing message specification (MMS), generic object oriented substation event (GOOSE), and sampled values (SV).

However, these protocols are not designed with security in mind and, thus, vulnerable to various cyberattacks [3], [4]. An example is CrashOverride [5], a malware capable of spoofing malicious IEC 61850 payloads to control substation equipment and shut down the power grid. Although the IEC 62351 security standard [6] exists to protect these protocols, those security measures (i.e., digital signatures and authentication schemes) are often not implemented in real-world systems due to legacy and performance issues [7]. Hence, many security practitioners use intrusion detection systems (IDSes) to analyze network traffic and detect cyberattacks [8], [9]. In this direction, many IDS solutions have been developed using the open-source Zeek framework [10]. Zeek relies on protocol parsers to perform deep packet inspection and custom scripts that operate on parsed data to generate intrusion alerts. While there are parsers to decode well-known ICS protocols such as DNP3 [11] and IEC-104 [12], none has been developed for IEC 61850-based protocols, in particular MMS. The reasons are mainly due to the complex data model (i.e., optional fields that may or may not exist in the packet), the need to support a large number of MMS services, and the recursive data structures. Furthermore, sending every MMS packet to Zeek for analysis may increase the processing time to respond to malicious traffic.

Motivated by these gaps, we design a protocol parser for Zeek to analyze MMS traffic in a power grid environment. The parser is developed using BinPAC [13] and can extract packet metadata which Zeek associates as events in the Zeek scripts to implement IDS rules to detect network anomalies. In more detail, we explain our design choices for handling optional fields and recursive MMS data structures. To reduce the processing load, we develop an automatic program to allow users to customize which MMS packets to send to Zeek for analysis. We consider different cases of optional fields in the MMS packets and design extensive test cases to prove the parser's correctness. We further conduct throughput and latency experiments to demonstrate the performance improvements of using custom events and provide sample Zeek scripts to explain how events are used to construct IDS rules. By open-sourcing the parser [14], [15], users can adapt it to design IDS solutions or use it to analyze attack datasets (e.g., [16]–[18]) for cybersecurity research.

II. Related Work

Many tools are available to interpret MMS traffic. For example, an MMS plugin is available for Wireshark to perform static analysis. The plugin is created using the Asn2wrs compiler [19]. There are also tools for generating MMS packets. An example is libiec61850 [20], which is available in C++ and Java. Another tool is IEDExplorer, which is an IEC 61850 tool for generating and testing MMS communications [21]. However, only limited services are supported, such as initiate/-conclude, read and write services. In [22], a communication stack library was developed for MMS using object-oriented approach. The library is developed to understand the MMS stack, including providing support for implementing MMS-based applications. We note that none of the tools can be

978-0-7381-3184-9/21/$31.00 ©2021 IEEE

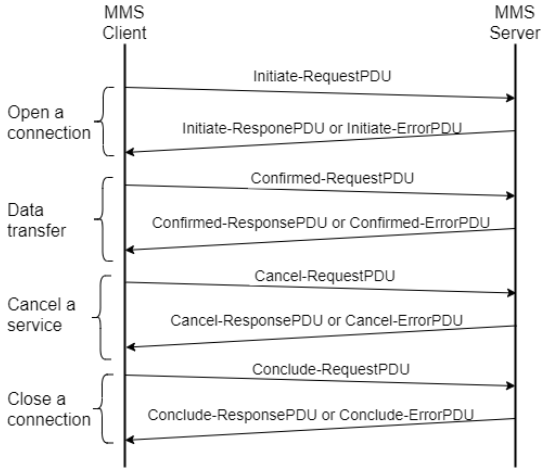

Fig. 1. MMS PDUs between a client and a server

integrated with IDS solutions in the open-source community. In contrast, we design and implement an MMS parser for Zeek using BinPAC to extract semantic information from packets as events, which Zeek then processes at the script level to analyze the traffic patterns for signs of attack.

III. Preliminaries

A. MMS Protocol Specifications

As shown in Fig. 1, MMS is a client-server protocol in which the client sends MMS protocol data units (PDUs) to request MMS services, while the server responds with the requested data. More specifically, the initiate-RequestPDU is used whenever the client wants to establish a connection with the server. The initiate-RequestPDU is confirmed by initiate-ResponsePDU from the server. If an error occurs during the connection, the server will reply with the initiate-ErrorPDU; otherwise the next phase is data transfer, which involves sending a confirmed-RequestPDU (client) and a confirmed-ResponsePDU (server). Each instance of the confirmed-RequestPDU and confirmed-ResponsePDU is related by an unique InvokeID integer. It is through confirmed-RequestPDU that MMS service messages are exchanged to transfer data. Some example of MMS services are getNameList, read, write, GetVariableAccessAttributes, etc. If the server cannot provide the requested data, the server will send a confirmed-ErrorPDU containing the service type and the InvokeID from the request PDU. At any point in time, the client can initiate a cancel-RequestPDU to cancel a specific service. The server will then follow up with a cancel-ResponsePDU if successful or cancel-ErrorPDU to indicate failure. Finally, to close a connection, the client sends a conclude-RequestPDU to the server. Similarly, the server replies conclude-ResponsePDU to confirm or conclude-ErrorPDU when an error occurs.

The MMS message structure is described using ASN.1 Notation and is encoded according to the Basic Encoding Rules (BER) [23]. ASN.1 uses keywords such as “SEQUENCE”, ”SEQUENCE OF”, “BOOLEAN”, IMPLICIT”, “OPTIONAL” to define the syntax of messages. For example, confirmed-RequestPDU is represented as a ”SEQUENCE”. Each keyword is associated with a tag identifier to identify its type. BER, on the other hand, specifies how data should be encoded for transmission. Each data is always encoded in tag-length-value (TLV) format. The first field refers to the ASN.1 identifier, the second field specifies the length of the value, and the third field represents the data value. Typically, the type is read first, followed by the length field to determine the number of bytes to serialize or deserialize the data.

B. Mapping of IEC 61850 to MMS

The approach to mapping IEC 61850 to MMS [24], [25] is to decompose a physical device into smaller functional units called logical devices (LD), logical nodes (LN), functional constraints (FC), data objects (DO), and data attributes (DA). An LD is a virtual representation of a physical device that describes a particular substation functionality such as control, protection, and measurement, whereas an LN is a subset of the functionality within a logical device. For example, a circuit breaker provides control functions and can be modeled as an XCBR logical node [26]. It contains various DO such as Loc to determine the operation mode, Pos to determine the position of the switch, etc. Each DO contains DA (e.g., stVal) to convey the current status of the device. FC, on the other hand, specify services (e.g., read, write) that are allowed to operate on the data attributes. These definitions make up the MMS object reference (e.g., IED1_CTRL/XCBRSTPos$stVal) to access the different states of the device.

The various data objects are specified in an IED capability description (ICD) file produced by the manufacturer. The ICD file describes the data model hierarchially in XML format to enable easy configuration for deployment. It comprises three sections: Communication, IED, and DataTypeTemplates. The communication tag defines the connections such as subnetworks, IP addresses, and access points. The IED tag describes the IED data model comprising the LD and their associated LN names. The datatype template defines the instantiated LN type, DO type, DA type, and enumerate datatype associated with each LD [27].

The IEC 61850 standard further defines a set of abstract communication services to operate on DO and DA. Example services are querying objects, getting/setting data values, controlling system objects, manipulating reports or logs, including other services like file operations. The abstract data and service models are then mapped to the MMS protocol based on associating MMS objects and MMS services to the various IEC 61850 objects and communication services. More details can be found in [2], [26].

C. Zeek and BinPAC

Zeek is a network security tool that provides deep packet inspection and intrusion detection capabilities. Zeek’s architecture is based on two components: an event engine and a policy script interpreter. The event engine parses packets from the network to generate events containing details of the packets. The events are then passed to the policy script interpreter

where Zeek scripts execute a set of security policies on the events to generate logs and raise notifications.

Zeek also comes preinstalled with BinPAC [13], a protocol parser generator. BinPAC allows users to extend Zeek's functionality by defining how a new protocol is parsed and how events are created. More specifically, BinPAC provides a set of template files — protocol.pac, analyzer.pac, and events.bif. For example, packet parsing is defined in the protocol.pac file. Events are defined in the analyzer.pac and events.bif files which then forward the details extracted by the protocol.pac parser to the policy script interpreter for analysis. The general workflow of using BinPAC is as follows: we define the packet structure, i.e., the type and the length of each packet field in the protocol.pac file. When a packet is parsed, the packet structure will be passed to the analyzer.pac for deserialization. Additionally, the analyzer.pac file implements callbacks to generate events that will be processed by Zeek scripts. The events must be defined in the events.bif file before they can be used by the analyzer.pac file. Finally, BinPAC provides a compiler toolchain to convert the files into corresponding C++ codes for integration with Zeek as a plugin.

IV. MMS Parser Design

A. Design Overview

Fig. 2 shows the proposed design and the testing workflow for verifying the MMS parser's correctness. The first step is to translate the MMS protocol specifications into BinPAC scripts. We define the various MMS PDU and MMS service packet structures as record types in the mms-protocol.pac file. Then, we specify the decoding logic in the mms-analyzer.pac file to deserialize the byte string data. Deserialization is based on BER-TLV encoding rules. In other words, we identify MMS messages based on their ASN.1 tag identifiers specified in [25]. For example, tag $0xa0$ represents an MMS confirmed-RequestPDU and tag $0xa1$ denotes the getNameList service. Next, we read the length to determine the size of the value field to deserialize. Finally, we extract and decode the value. For more complex packet structures, we define additional parsing logic in the analyzer file to process them More details are described in Section IV-B.

On successful parsing of a packet, the analyzer file will generate events and store them in the event queue for processing by the Zeek scripts. Depending on the type of analysis, the script either logs the information for monitoring purposes or raises alerts when a packet violates certain security policies. In our design, we distinguish between two types of events: (1) generic events and (2) custom events. Generic events are generated based on MMS service types (i.e., getNameList, read, write, etc.). Custom events are generated based on user-defined MMS object references. We introduce custom events as a means to reduce the number of events that Zeek needs to process in order to improve the response time to critical events. Another reason is to provide users with more fine-grained control to prioritize events monitoring.

To generate custom events, we develop a Java program (EventGenerator.java) that takes a configuration file as input

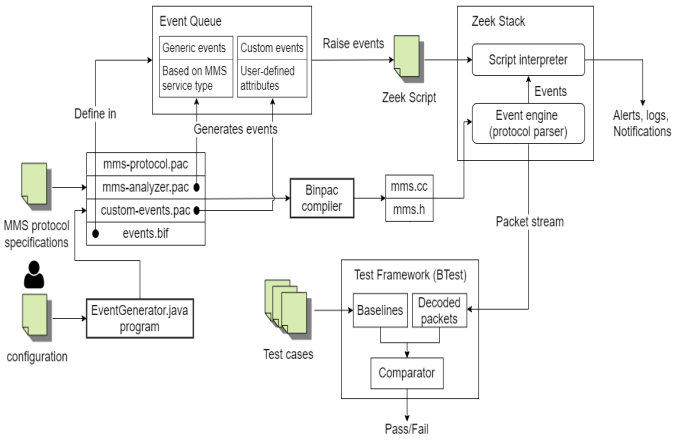

Fig. 2. Building blocks of the MMS parser design. The parsing and decoding logic are defined in *.pac scripts, which will be compiled by BinPAC into C++ code to be deployed on Zeek's event engine. Sample MMS test cases are fed into the BTest framework as baselines where they will be compared with decoded packets from the Zeek's event engine to output pass/fail.

to automatically generate a custom-events.pac file. The configuration file contains user-specified MMS object references. Based on that information, the automatic program applies text filters on the parsed MMS data to generate custom events for analysis. We provide more details in Section IV-C. As a last step, we formally define the events in the events.bif file so that mms-analyzer.pac and custom-events.pac files can reference them for execution. Once the scripts are implemented, BinPAC compiles the scripts into C++ codes and integrate them as a plugin into Zeek's event engine. To verify the protocol parser's correctness, we generate some test cases as baselines and use the BTest framework to verify that the MMS packets are decoded correctly.

B. Parsing Strategies

As mentioned previously, the structure of MMS packets is complex and of variable length. For example, a packet may contain an OPTIONAL field, which means that the specified field may or may not exist. If the OPTIONAL field is not present, parsing will fail and the packet will be decoded wrongly. In addition, some MMS packets may contain multiple nested TLV structures, which further complicate the parsing logic. Such dynamic data structure cannot be fully described in the mms-protocol.pac file, so different parsing strategies need to be adopted in the mms-analyzer.pac file. In the following, we use the GetVariableAccessAttributes-Response packet in Fig. 3 as an example to enumerate the different cases and explain how to overcome them.

Case 1: Optional field followed by a required field: The address field in the GetVariableAccessAttributes-Response packet is an OPTIONAL field followed by typeDescription, which is required. As there is no 'peek' function in BinPAC to check the next byte, the solution is to read the next field and set a Boolean flag to true when the tag of the OPTIONAL field (i.e., address field) is found, and then use ASN1OptionalEncodingMeta provided in mms-asn1.pac file to

```
GetVariableAccessAttributes-Response::= SEQUENCE{
 mmsDeletable [0] IMPLICIT BOOLEAN,
 address [1] Address OPTIONAL,
 typeDescription [2] TypeDescription,
 accessControlList [3] CHOICE {
   basic BasicIdentifier,
   extended ExtendedIdentifier
 } OPTIONAL,
 meaning [4] ObjectName OPTIONAL
 }
TypeDescription::= CHOICE {
 array [1] IMPLICIT SEQUENCE {
   packed [0] IMPLICIT BOOLEAN DEFAULT FALSE,
   numberOfElements [1] IMPLICIT Unsigned32,
   elementType [2] TypeSpecification
 },
 structure [2] IMPLICIT SEQUENCE {
   packed [0] IMPLICIT BOOLEAN DEFAULT FALSE,
   components [1] IMPLICIT SEQUENCE OF SEQUENCE {
     componentName [0] IMPLICIT Identifier OPTIONAL,
     componentType [1] TypeSpecification
 ...
}
```

Fig. 3. Snippet of a GetVariableAccessAttributes-Response packet

parse further if needed. If the Boolean flag is false, it means that there is no OPTIONAL field.

Case 2: Consecutive OPTIONAL fields: In GetVariableAccessAttributes-Response packet, accessControlList and meaning are two consecutive OPTIONAL fields. So there are 2^2 possible combinations to encode those OPTIONAL fields in an MMS message. There is also a possibility that none of the OPTIONAL fields is present in the encoded MMS message. That means there are no more bytes to parse after the last required field. To address consecutive OPTIONAL fields, a "case" compositor is used to switch between cases when OPTIONAL field(s) are present or otherwise. If at least one of the OPTIONAL fields is present, more bytes need to be parsed, and hence the MMS message will have non-zero length. Based on this idea, we define a function in mms-analyzer.pac to check the message length. If the remaining length is not zero, the mms-analyzer.pac will pass the processing to the mms-protocol.pac to continue with the parsing of OPTIONAL fields. The parsing terminates only when the function in mms-analyzer.pac returns false, indicating no more data is left to be parsed. However, this approach can only handle two to three consecutive OPTIONAL fields in the mms-protocol.pac file. For MMS services with up to seven successive OPTIONAL fields (e.g., DefineAccessControlList-Request), the parsing logic can get complicated and difficult to maintain.

Case 3: Recursive calling problem: The GetVariableAccess Attributes-Response service contains TypeDescription, which has a recursive call for TypeSpecification. Without knowing how many recursive structures are there in a packet, it is difficult to define the packet structure in mms-protocol.pac. Parsing such dynamic packets will result in a circular bug [28]. To resolve this error, we shift the processing to the mms-analyzer.pac and decode each TLV structure separately.

C. Generic and Custom Event Generation

The next step is to generate events for Zeek to analyze. When events are triggered, they will be processed by event handlers defined in the Zeek script. In the following, we elaborate on the generation of generic and custom events.

1) Generic Events: We generate generic events by creating an event for each type of MMS service packet. As each MMS packet is parsed, we store the values and their data types into two separate vectors. We store the data types because they are required to interpret the data values and to perform computational analysis at the script level. After all the data items have been processed, we define an event and pass the two vectors as arguments to the script interpreter. Below is how we instantiate an event for the status-request service in the mms-analyzer.pac file. When the event fires, the connection information, InvokeID, the data values, and the corresponding data types will be sent to the script layer for processing.

```
BifEvent::generate_status_request(
connection()->bro_analyzer(),
connection()->bro_analyzer()->Conn(),
invoke_id,
vec_data,
vec_datatype);
```

Some MMS services may contain domainID and itemID information. Examples are write-request/response and read-request/response services. DominID refers to the LD name, while itemID represents the MMS object reference, i.e., $LN\$FC\$DO\$DA$. They are present in the packet to uniquely identify messages from different devices. To process those packets, we concatenate the domainID and itemID before storing them as a string into a global map. The map serves to associate the value to a key. The key is the InvokeID which is unique for each MMS request-response pair. On processing a response packet containing data values, the InvokeID will be used to retrieve the stored value (i.e., concatenated domainID_itemID string) from the map. This information, along with the data values and data types from the response packet will be passed to the script layer. Below is the syntax for defining a GetNameList-response event with domainID and itemID information.

```
BifEvent::generate_get_name_list_response(
connection()->bro_analyzer(),
connection()->bro_analyzer()->Conn(),
invoke_id,
string_to_val(concatenated_domain_itemid),
vec_data,
vec_datatype);
```

After that, we declare event handlers in the events.bif file to interface with the Zeek script. The handlers for the above two events are defined as follows:

```
event status_request%(c: connection,
invoke_id: count,
data: string_vec,
datatype: index_vec%);

event get_name_list_response%(
c: connection,
invoke_id: count,
identifier: string,
data: string_vec,
datatype: index_vec%);
```

where c is the connection information, InvokeID is an unique identifier associating the request and response packets, identifier is the concatenation of the domainID and itemID.

2) Custom Events: Generic events can be quite resource-intensive since we are generating an event for each MMS service type. Besides, most service types often include packets from other devices that are not needed for analysis. While it is possible to filter unwanted events in the Zeek script, script processing is slow and consumes many CPU resources. Therefore, we decouple the event generation logic from the mms-analyzer.pac file and create custom events instead.

Our approach is to filter parsed packets based on matching the domainID and itemID information provided by the users. In other words, we utilize the global map maintained by the mms-analyzer.pac file to determine if we should send the data values to the script as events. To do that, we develop an EventGenerator.java program to automate the generation of a custom-events.pac file. The syntax for creating custom events is similar to generic events except that we populate logic in the custom-events.pac file to filter specific packets for event monitoring. In more detail, we leverage the ICD file to enable users to extract the MMS object references they wish to monitor. The ICD file contains private blocks that define the mappings between the device's local variables and the MMS object references. The private blocks are generated by the process owners before device commissioning and can be enabled via the export function of the device configuration tool. As shown in Fig. 4, the device variable name is embedded in one of the DA as private property. Using that information, we developed a SCLParser.java program to parse the ICD file and extract the mappings into a CSV file. Specifically, our program reverses the path starting from the DA level where the variable name resides to the LD level and re-constructs the MMS object reference. The output of parsing the ICD is the following: device_variable_name, domainID_itemID (i.e., the concatenation of the device name, LD name, and MMS object reference — $LN\$FC\$DO\$DA$), and the index position of the MMS object reference. The index is needed to access the attribute value in the vec_data vector created by the mms-analyzer.pac. Based on this information, the users can choose the variables they want to monitor and pass the corresponding domainID_itemID in a configuration file to the EventGenerator.java program. The program then implements the pseudocode below to generate the custom-events.pac file,

```
function rule_function():bool %{
  if(StringMatch(X,<string1>)==True){
     value = vec_data[<index>];
     BifEvent::generate_<variable_name>(
     connection()->bro_analyzer(),
     connection()->bro_analyzer()->Conn(),
     service,
     invoke_id,
     value);
  }
  return false;%}
```

where $StringMatch$ is a string matching operation, X is the concatenated domainID_itemID retrieved from the global map

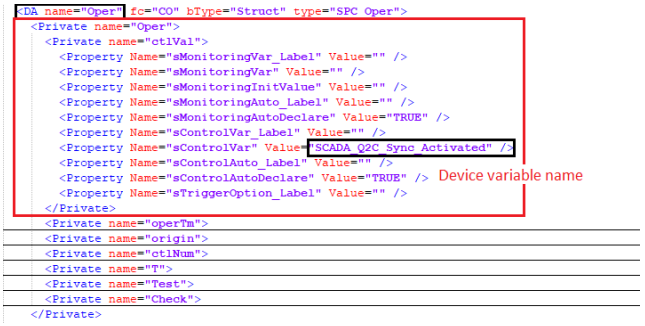

Fig. 4. Mapping of device local variable to the DA object in the private block. Sensitive details are intentionally omitted for privacy reasons.

by the mms-analyzer.pac and $< string1 >$ is the domainID _itemID specified by the user. On a successful match, the program will retrieve the attribute value from the vec_data vector by $index$, create a BifEvent and send the value along with other connection information to the script. The program will replicate many BifEvent functions equal to the number of domainID_itemID pairs in the configuration file. Each BifEvent is indexed by $< variable\_name >$ for easy identification. We also send the service type encoded in integer format to the script to identify the type of service message being raised. The integer-to-service type mapping is defined in the mms-tags.pac file (available in our Github [14]).

Next, we define a function call with the same name (i.e., *rule_function()*) in the mms-analyzer.pac file so that each time a packet is processed, it invokes the custom-events.pac file. To summarize, the user specifies the domainID_itemID information in the configuration file in CSV format. The EventGenerator.java program then auto-populates the skeleton code with information from the configuration file to create custom events. The Java programs can be found in our Github [15]. The benefit of using a separate custom-events file is that users can customize events without touching the base code. Furthermore, it offers greater flexibility, reduced complexity, and improved real-time performance.

V. Evaluation

In this section, we explain the experimental setup and evaluate the correctness and performance of our parser. Our setup consists of an industrial PC, a laptop, and a Hirschmann switch. Both industrial PC and laptop are connected to the Hirschmann switch. The industrial PC runs Zeek, preinstalled with our MMS parser, and acts as the IDS device. The laptop is used as a traffic generator to send MMS packets into the network. We set up the IDS to passively monitor the network traffic by configuring the switch port connected to the laptop as the mirror port so that packets sent by the laptop are replicated to the industrial PC for analysis by Zeek.

A. BTest Framework

We used BTest framework provided by Zeek as a separate installation to verify the correctness of our parser. The input to BTest is a set of testing pcaps, most of which were generated using libIEC61850 v1.4.0 [20] while some were extracted

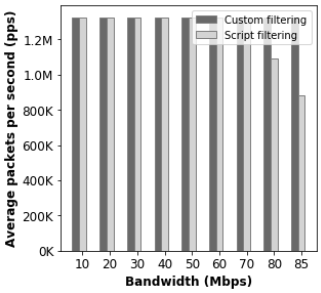

Fig. 5. Throughput (pps) for one flow

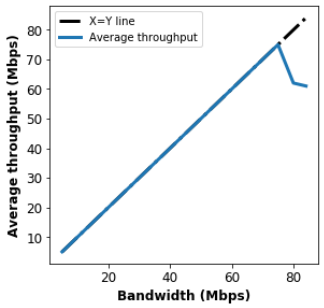

Fig. 6. Overall throughput for 4 flows

from pcaps generated in SUTD's EPIC lab [29]. For each pcap, we extracted all the fields and values to establish the baselines. Next, we run btest-diff command on the same set of pcap files to check if the output matches the baseline. If the output matches, btest-diff will return success, else it returns failure. We have validated that our parser can decode six MMS PDUs and fifteen MMS confirmed services correctly. A complete list of verified MMS services is available on our Github [14]. Test cases and corresponding pcaps are also included in our repository for reproducibility.

B. Throughput and Latency

We generated a large pcap file (∼8.5GB) comprising 4 flows with the majority of the packets using MMS read service (88.4%) and some using MMS write service (0.3%). The rest are non-MMS packets. The throughput measurements were recorded by using tcprelay to replay packets at bandwidths from 10Mbps to 100Mbps. For each bandwidth, we performed 10 runs to measure the average throughput. All experiments were conducted on the industrial PC running 4 cores with 2.8GHz processors and 16GB RAM.

In the first experiment, we measured the throughput of one flow in packets per second (pps) under two different settings. In the first setting, we implemented some filtering logic (i.e., using domainID_itemID) in a Zeek script to process MMS response packets from a specific device. In the second setting, we generated a custom-events.pac file to filter MMS response packets from the same device at the event engine level. As shown in Fig. 5, the parser starts dropping packets beyond 80Mbps when the filtering is performed at the script. This degradation is because the event engine generates an event for every packet that is parsed, which slows down the processing speed of the script layer. With custom filtering at the event engine, only selected packets will be passed to the Zeek script. Thus, the parser can process all packets without any packet drop. Fig. 6 measures the overall average throughput for all flows without custom filtering. The figure shows that the throughput peaks at around 75Mbps, which is sufficient for most MMS-based applications involving hundreds of devices because each MMS device has a low traffic volume of 1-2 packets per second. Even when operating at this maximum throughput, we can use custom filtering to prioritize events to the script layer to avoid missing any critical packets for intrusion analysis.

TABLE I
AVERAGE LATENCY FOR PROCESSING A PACKET

	Event Engine (μs)	Script (μs)
Base pcap (4 flows)	157.94	202.97
Script filtering (1 flow)	139.30	156.12
Custom filtering (1 flow)	125.03	140.55

In terms of latency, we measured — (1) the time required for the parser to decode the packet and trigger an event, and (2) the cumulative time for the packet to reach the script level for processing. In the latter, the latency measured at the script level includes the processing delay incurred by the event engine. In all experiments, the packets were injected into the network at a low bandwidth (0.1Mbps) and the latency was averaged out over 10 runs. As shown in Table I, the latency per packet is the highest for multiple flows because the parser processes every flow and every packet, and some packets are larger in size. Adding a simple script to process all the generated events results in an additional delay of 45μs per packet. If we consider one flow with script filtering, the latency is lower, particularly 139.3μs at the event engine and 156.12μs at the script layer. Compared to script filtering, custom filtering has the lowest latency because less events are generated. Although the difference is small, about 15.6μs per packet, the latency savings can be substantial if we consider many packets to process at the script. Lower latency means Zeek can process more packets. The latency is also directly proportional to the complexity of the script. If the script is complex, the latency per packet will increase further.

C. Attack Detection Case Study

For completeness, we provide sample Zeek scripts to show how events can be used to detect attacks. We further evaluate the "mean time to detect" metric, i.e., the time between attack and detection, to demonstrate the performance gain of using custom events for intrusion detection. The case study used here is the generator synchronization process in EPIC [29], [30]. Due to space limitations, we refer readers to our Github [15] for more details. The key MMS messages in the synchronization process are start_sync command, phase angle information, and sync_complete command. We assume an attack scenario where the incoming generator is rotating at the same speed as the reference generator. Thus, the phase angle will not decrease at the normal rate of 4-5° and the circuit breaker will not close to connect the generator. Based on this attack scenario, we implemented two Zeek scripts (available on Github [15]) to raise alarms if the phase angle change is outside the predefined limits of 4-5°. The first script uses generic events, i.e., MMS read response and MMS write request, which requires filtering for the MMS object references to identify the start_sync command and the phase angle packets. On receiving the start_sync command, the script checks whether the phase angle is decreasing at a normal rate. Otherwise, the script issues an alarm. The second script

implements custom filtering, where the corresponding MMS object references are filtered in the event engine and processed directly by the custom handlers in the script. As such, no filtering is required at the script level. Our results show that both scripts can detect attacks. However, compared to the generic script that requires 6.2μs, custom filtering only takes 2.6μs to detect an attack.

VI. Conclusions

This paper presents the design of an MMS parser, which can be integrated with Zeek directly to analyze MMS traffic in a power grid environment. We discuss the challenges of parsing MMS packets and propose workarounds in BinPAC to support parsing dynamic data structures. We provide customizations through separate Java programs to allow users to personalize events generation for best performance. The parser was evaluated for correctness using the BTest framework. Experiments were conducted to stress-test the parser's performance. The throughput results showed that the parser can achieve a high throughput with reasonable delay. Specifically, the parser can reach a throughput of 75Mbps. By open-sourcing the parser, security practitioners and researchers can better understand MMS semantics and design IDS rules to identify cyberattacks.

Acknowledgment

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore in part under its Energy Programme and administrated by the Energy Market Authority (EP Award No. NRF2017EWT-EP003-047), in part under its National Satellite of Excellence DeST-SCI programme (Award No. NSoE_DeST-SCI2019-0008), and in part under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[1] Q. Song, W. Sheng, L. Kou, D. Zhao, Z. Wu, H. Fang, and X. Zhao, "Smart substation integration technology and its application in distribution power grid," *CSEE Journal of Power and Energy Systems*, vol. 2, no. 4, pp. 31–36, 2016.

[2] "IEC 61850 - Communication networks and systems for power utility automation - ALL PARTS," Feb 2020. [Online]. Available: https://webstore.iec.ch/publication/6028

[3] R. Zhu, C.-C. Liu, J. Hong, and J. Wang, "Intrusion detection against MMS-based measurement attacks at digital substations," *IEEE Access*, 2020.

[4] B. Kang, P. Maynard, K. McLaughlin, S. Sezer, F. Andrén, C. Seitl, F. Kupzog, and T. Strasser, "Investigating cyber-physical attacks against IEC 61850 photovoltaic inverter installations," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2015, pp. 1–8.

[5] J. Slowik, "Crashoverride: Reassessing the 2016 ukraine electric power event as a protection-focused attack," *Dragos, Inc*, 2019.

[6] "IEC 62351:2020 - Power systems management and associated information exchange - Data and communications security - ALL PARTS," May 2020. [Online]. Available: https://webstore.iec.ch/publication/6912

[7] H. C. Tan, C. Cheh, B. Chen, and D. Mashima, "Tabulating cybersecurity solutions for substations: Towards pragmatic design and planning," in *2019 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, 2019, pp. 1018–1023.

[8] P. I. Radoglou-Grammatikis and P. G. Sarigiannidis, "Securing the smart grid: A comprehensive compilation of intrusion detection and prevention systems," *IEEE Access*, vol. 7, pp. 46 595–46 620, 2019.

[9] Y. Hu, A. Yang, H. Li, Y. Sun, and L. Sun, "A survey of intrusion detection on industrial control systems," *International Journal of Distributed Sensor Networks*, vol. 14, no. 8, p. 1550147718794615, 2018.

[10] "Zeek: An Open Source Network Security Monitoring Tool," accessed 2020-08-20. [Online]. Available: https://zeek.org/

[11] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. K. Iyer, "Adapting Bro into SCADA: Building a specification-based intrusion detection system for the DNP3 protocol," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, 2013, pp. 1–4.

[12] J. Chromik, A. Remke, B. R. Haverkort, and G. Geist, "A parser for deep packet inspection of IEC-104: A practical solution for industrial applications," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks – Industry Track*, 2019, pp. 5–8.

[13] R. Pang, V. Paxson, R. Sommer, and L. Peterson, "Binpac: A yacc for writing application protocol parsers," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 2006, pp. 289–300.

[14] "MMS parser protocol parser for Zeek IDS," September 2021. [Online]. Available: https://github.com/smartgridadsc/MMS-protocol-parser-for-Zeek-IDS

[15] "Custom MMS event generator," September 2021. [Online]. Available: https://github.com/smartgridadsc/mms-custom-event-generator

[16] D. Mashima, B. Chen, P. Gunathilaka, and E. L. Tjiong, "Towards a grid-wide, high-fidelity electrical substation honeynet," in *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2017, pp. 89–95.

[17] P. P. Biswas, H. C. Tan, Q. Zhu, Y. Li, D. Mashima, and B. Chen, "A synthesized dataset for cybersecurity study of IEC 61850 based substation," in *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2019, pp. 1–7.

[18] P. P. Biswas, Y. Li, H. Chuan Tan, D. Mashima, and B. Chen, "An attack-trace generating toolchain for cybersecurity study of IEC 61850 based substations," in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020, pp. 1–7.

[19] "Chapter 14. Creating ASN.1 Dissectors. Part II Wireshark Development," Jan 2021. [Online]. Available: https://www.wireshark.org/docs/wsdg_html_chunked/CreatingAsn1Dissectors.html

[20] "LibIEC61850/Lib60870-5: Open Source Libraries for IEC 61850," April 2019. [Online]. Available: https://libiec61850.com/libiec61850/about/

[21] "IEDExplorer: IEC61850 IED Explorer in .net," August 2020. [Online]. Available: https://sourceforge.net/projects/iedexplorer/

[22] J. Park, E. In, S. Ahn, C. Jang, and J. Chong, "IEC 61850 standard based MMS communication stack design using OOP," in *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2012, pp. 329–332.

[23] H. Falk and M. Burns, "MMS and ASN. 1 encodings," *Systems Integration Specialists Company, Inc.(SISCO), USA*, 1996.

[24] "ISO 9506-1, Industrial automation systems — Manufacturing Message Specification — Part 1: Service definition," 2003. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:37079:en

[25] "ISO 9506-2, Industrial automation systems — Manufacturing Message Specification — Part 2: Protocol specification," 2003. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9506:-2:ed-2:v1:en

[26] P. Matoušek, "Description of IEC 61850 communication," in *Technical Report*. Brno University of Technology, 2018.

[27] B. Jang, A. Abubakari, N. Kim *et al.*, "IEC 61850 SCL validation using UML model in modern digital substation," *Smart Grid and Renewable Energy*, vol. 9, no. 08, p. 127, 2018.

[28] "BinPAC segfaults on circular record dependencies," May 2016. [Online]. Available: https://bro-tracker.atlassian.net/browse/BIT-1596

[29] S. Adepu, N. K. Kandasamy, and A. Mathur, "EPIC: An electric power testbed for research and training in cyber physical systems security," in *Computer Security*. Springer, 2018, pp. 37–52.

[30] A. Siddiqi, N. O. Tippenhauer, D. Mashima, and B. Chen, "On practical threat scenario testing in an electric power ics testbed," in *Proceedings of the 4th ACM Workshop on Cyber-Physical System Security*, 2018, pp. 15–21.