Validating and Securing DLMS/COSEM Implementations with the ValiDLMS Framework

Henrique Mendes, Ibéria Medeiros, Nuno Neves
LASIGE, Faculdade de Ciencias, Universidade de Lisboa, Portugal
mendes.california@gmail.com, {imedeiros, nuno}@di.fc.ul.pt

***Abstract*—The electrical grid is a critical infrastructure for modern society. It has been evolving into a smart(er) grid, allowing infrastructure aware decisions based on data collected in real-time from smart meters and other devices. Smart meters and their uplinks have, however, limited physical security due to their location within customer premises. DLMS/COSEM is a standard protocol for remote interactions with smart meters, often being deployed above power-line communication links. The paper presents the ValiDLMS framework, the first open source solution for validation and security auditing of DLMS/COSEM implementations using this communication profile. The framework was developed as an extension to Wireshark and was used to analyse an industry partner's DLMS/COSEM implementation. The results show that ValiDLMS can effectively support the discovery of bugs and/or other non-conformance problems.**

I. Introduction

The power grid is a critical infrastructure for society as without it many vital sectors would cease normal operation (e.g., telecommunications, transportation). Over the past decade, the technology used in the infrastructure has seen gradual advances, enabling the *smart grid* transformation and bringing the promise of increased functionality and better quality of power delivery. One of the major evolutions is occurring with the new advanced metering infrastructure (AMI), which allows a two-way communication between the operator stations and the meters at the customers' premises. This enables the access to on-demand information, such as various usage statistics, load profile, outage logs, and tamper notifications. In particular, *smart meters* are letting utilities (and customers) make grid-aware decisions based on data collected in real-time.

However, the current deployment of smart meters raises scepticism and concerns because of potential security issues. If vulnerabilities are found in meters after their deployment, this could lead to various kinds of problems as result of an attack (e.g., fraud on meter readings). In addition, of particular concern are the networks connecting meters to the grid, as uplinks have limited physical security due to their location within the customers' sites. These uplinks can be easily observed and tampered with by an adversary, giving opportunity for attacks that for example modify the commands that are issued by the operator to the local meter [1]. This communication layer could also enable remote attacks to be carried out, namely to other meters in the same region and the operator stations, allowing a rise in scale and greater negative effects — e.g., they could lead to mass privacy breaches or a localized power outage. Therefore, it is fundamental to ensure proper configuration and management of the mechanisms employed in the AMI network.

The Device Language Message Specification and Companion Specification for Energy Metering (DLMS/COSEM) is a standard protocol for remote interaction in the smart grid, often being deployed above power-line communication (PLC) links. It is becoming the most favoured choice within smart grid protocols due to its energy-type-agnostic interface model and its clear separation between that interface model and the communication layers beneath it. In addition, it offers several other benefits and features often not available in alternative protocols, such as encryption and digital signatures [3], [4].

Even though the DLMS/COSEM protocol has already been included in the regulations of some countries [5], there is little amount of work examining the DLMS/COSEM's deployments and ascertaining the effectiveness of its security features. The paper addresses this gap by proposing the ValiDLMS framework, the first open source solution for the validation and security auditing of DLMS/COSEM implementations using a PLC profile. The framework takes as input the messages exchanged among the devices, for example the packets sent between a data concentrator (DTC) and the smart meters, and then it verifies if the format and contents of the messages appear as expected. ValiDLMS also looks for several classes of vulnerabilities that have been identified in the past, starting with the selected cipher suite and methods for authentication. The framework was developed as an extension of Wireshark [6] and was used to analyze an industry partner's DLMS/COSEM implementation. The results show that ValiDLMS can discover a few security problems, such as transmitting clear passwords, and/or other non-conformance problems.

The contributions of the paper are: (1) the design of the ValiDLMS framework for DLMS/COSEM validation and security auditing; (2) an experimental evaluation aiming at analysing a DLMS/COSEM implementation currently being tested for a country wide deployment.

II. DLMS/COSEM Protocol

The DLMS/COSEM protocol is used for energy metering. DLMS models the communication entities and defines how information is formatted, transported and processed. COSEM specifies the COSEM interface classes (IC), the Object Identification System (OBIS), and the use of interface objects to model functions for meters [2], [10]. Also, the protocol defines

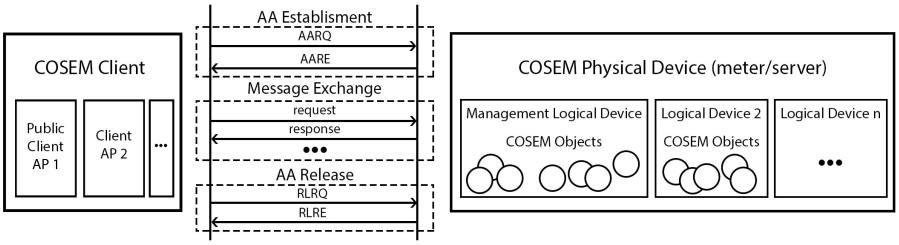

Fig. 1: DLMS/COSEM simplified model and communication pattern.

communication profiles which a client can connect to a server (e.g., TCP-UDP/IP, S-FSK PLC).

In DLMS/COSEM, a physical device (e.g., meter) holds one or more logical devices and each logical device contains a set of COSEM objects (instances of ICs), and is bound to a Service Access Point (SAP). SAPs depend on the layers below the COSEM application layer, and therefore they are determined by the communication profile being used. Fig. 1 depicts a simplified model and communication with DLMS/-COSEM. Application associations (AA) are established using their respective request (RQ) and response (RE) messages. After the message exchange the connection may be released (RL). The DLSM/COSEM communication established in an AA may be encrypted depending on the security settings agreed upon. DLMS messages are exchanged as encoded Application Protocol Data Unit (APDU).

During the AA establishment phase, the server and the client may authenticate with each other. DLMS/COSEM supports three kinds of authentication mechanisms, from no authentication to mutual. Also, it defines three security suites for cryptographic protection, each specifying the algorithms for authenticated encryption, key agreement, digital signature, hashing, key transport, and compression.

A. DLMS/COSEM weaknesses

This section presents some of the relevant vulnerabilities found in the DLMS/COSEM protocol (mostly exposed by Weith [12]), and the weaknesses that are still not addressed in the latest revision of the standard.

1) Weak authentication and cryptographic methods: The security header is sent in plaintext even in protected APDUs, which makes it prone to modification. By altering one bit in the security header, removing the last twelve bytes of an APDU, and updating its length, an attacker can transform an authenticated and encrypted APDU into an encrypted only APDU. If the security policy of the target device allows encrypted only APDUs, the attacker can downgrade the protection applied in APDUs, effectively causing a downgrade attack.

2) Information leakage: Some xDLMS services use service-specific encryption that results in a variant of the normal xDLMS APDU. In this case, the tag that identifies the APDU type is left in plaintext, revealing which xDLMS service is being transported in the APDU. This makes it trivial for an adversary to know when a get-request has been sent, regardless of the APDU protection. This information could then be used to determine how to decrypt the APDU, eventually with the brute force attack.

Another vulnerability is related with the fixed message sizes that can also help an adversary to find out what messages are being transmitted. This problem is particularly acute for APDUs with a fixed size, which is unique among the exchanged messages. This facilitates the guessing of the content in the fields of those messages, which in turn is useful to perform known-plaintext attacks.

In addition, a lot of known plaintext can be gathered from other sources, such as the user-information fields in AARQ responses. For example, it was found that HLS 5 (GMAC) authentication responses were the most reliable source of known plaintext, providing 22 bytes of easily obtainable information [12].

3) Message flow integrity: There is nothing included in the messages that allows a client to link the transmitted request with the arriving response. Therefore, the client cannot differentiate between the real response and a valid response introduced by a man-in-the-middle attack.

III. The ValiDLMS Framework

There are currently no known open-source tools for analyzing DLMS/COSEM communications over PLC. We propose ValiDLMS as an approach to validate DLMS/COSEM implementations and audit their security. The framework includes a number of modules, providing the capability for parsing and validating DLMS/COSEM messages, and the security analysis by employing fuzzing techniques and vulnerability tests.

A. Framework overview

The architecture of ValiDLMS is displayed in Fig. 2. It is divided in three layers: *DLMS/COSEM environment*, *interaction*, and *testing*.

In a typical deployment scenario of DLMS/COSEM, a client data concentrator (DTC) communicates with the smart meter servers to collect and/or set local information. One would like to determine if the exchanged messages (DLMS/COSEM environment layer) follow the specification of the protocol and if they are properly secured (Testing layer). The interaction layer interconnects these two layers, allowing the transmission of data between them. This means that, on one hand, the interaction layer captures the DLMS/COSEM communications occurring in the environment, and then delivers them to the testing layer. Here, they are analyzed by a tool (see Section IV) to determine if the messages conform with the protocol

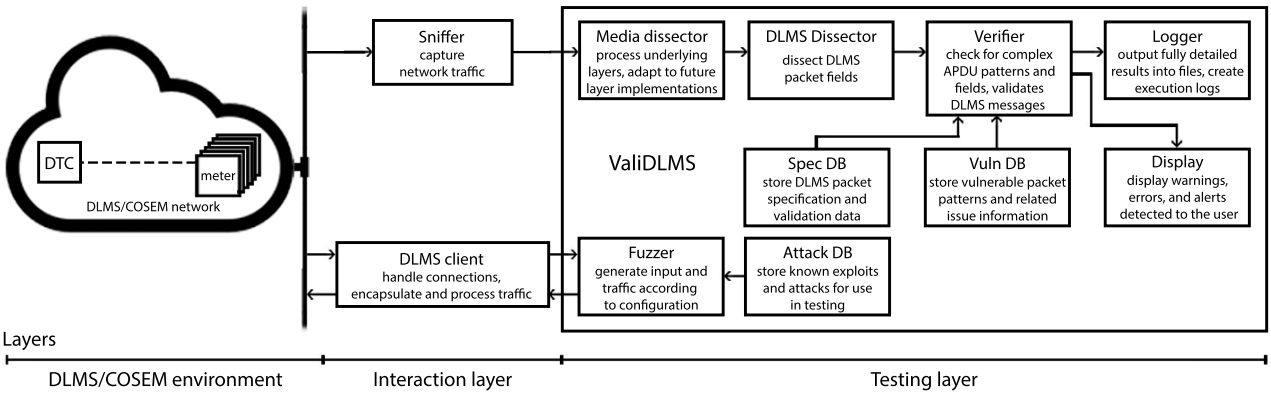

Fig. 2: ValiDLMS framework architecture.

specification and are protected from attacks. On the other hand, the interaction layer is also used as a means for the tool to perform security tests against the DLMS/COSEM environment, by carrying out attacks via packet injection with a fuzzer. Furthermore, attacks executed by the tool should also be captured and any side effect should be detected, issuing a warning if security deviations are found.

B. Interaction layer

The interaction layer is composed by two components – the *DLMS client* and the *sniffer* – that support the communications between the tool and the DLMS/COSEM network.

1) DLMS Client: The DLMS client component enables a two way communication between the fuzzing module of the tool and the DLMS/COSEM environment. It manages all the necessary connections, allowing the tool to take part in complex interactions with a DLMS/COSEM network, letting the fuzzing module to perform multi-stage attacks and other sophisticated malicious actions. It is also in charge of managing authentication and encryption credentials and certificates. The client should be able to encode and encapsulate DLMS/-COSEM traffic, as well as support the underlying media layers and communication profiles needed. The traffic generated by this component should be indistinguishable from traffic produced by a legitimate DLMS/COSEM client or server.

2) Sniffer: The sniffer component captures all the traffic from the environment being tested, and delivers it to the verification module of the tool, allowing a complete and in-depth analysis of the DLMS/COSEM messages. The sniffer gets a copy of the normal traffic as well as the testing messages inserted by the fuzzing module. Thus, through this component, ValiDLMS achieves the best coverage possible of the environment. Additionally, it can help in debugging tasks during the development of the fuzzing module.

C. Testing layer

The testing layer comprises components that implement fuzzing and verification modules. These components are detailed in Section IV. In the fuzzing module, a *fuzzer* component uses an *attack database* (attack DB in the Fig. 2) to run exploits and attacks, or otherwise induce the system to behave in an erroneous manner, while modifying packets with semi-random inputs and mutations. On the other hand, in the verification module, any traffic captured is dissected with two different parsers – *media dissector* and *DLMS dissector* – in resemblance with the DLMS/COSEM specification. The *verifier* component does most of the heavy lifting in the tool: it takes validation data from the *specification database* (Spec DB in the figure), and vulnerability data from the *vulnerability database* (Vuln DB in the figure), and checks: (1) if the DLMS implementation is in accordance with the specification; and (2) if there are exploitable vulnerabilities visible within and between the captured APDUs. If any deviation in specification or security violation is detected, the module outputs and logs a message, respectively, on the *display* and *logger* components.

IV. Tool components

This section looks at each component of the tool in more detail, highlighting specific aspects that are relevant for their implementation. Their connections are represented in Fig. 2.

A. Fuzzing module

Fuzzing can be useful when assessing the security of a system, due to its ability to automate the testing with semi-random data. This allows a greater number of tests to be generated and performed in a relatively fast and thorough manner. Random data is injected in order to check if it triggers some bug present in the system [13]. So, fault discovery can be accomplished by monitoring the system for different forms of misbehaviour, e.g., unexpected response messages [14]. There are two main fuzzing methods that we can employ depending on the method of generating data for injection: mutation-based fuzzing and generation-based fuzzing. The former modifies (mutates) pre-generated packets, whereas the latter uses input models to generate new packets using semi-randomness [16].

The fuzzing module is composed by two components: *attack database* and *fuzzer*, which are detailed bellow.

1) Attack database: The attack database stores attacks and tests cases for use by the fuzzer. The users can define which messages and patterns are to be employed in order to induce errors and reveal bugs in the system (e.g., either the DTC or

```
messageReplace(meterA)
{
m1 < meterA.getCipheredMessage(predictedMsg)
m1.removeAuthentication()
k1 < recoverKeystream(m1,predictedMsgs)
i1 < recoverIV(m1)
m2 < forgeVulnerableMsg(m1,cgf)
sendForgedMsgAs(meterA,m2,k1,i1)
}
{"Replaces a message with forged one using the tool's
global config. Cipher-only APDUs need to be accepted
and needs information on predicted messages"}
```

Fig. 3: Attack DB entry example.

the meter). Specific exploits can be added as building blocks for larger and multi-stage attacks.

Attack entries in the database have a name for referencing and a description, including limitations or caveats the attack depends on. The description will assist the user to understand the test, and also hint what the problem might be when the attack is successful. Fig. 3 exemplifies an attack that intercepts a message sent by a meter A, removes its authentication, recovers the keystream and invocation vector, forges a message based on the tool's configuration, and then replaces the original message by the forged introducing it in the network.

2) Fuzzer: The fuzzer component reads the tests from the attack database and additionally may generate semi-random input for message fields, or deliberately generate and send incorrect messages. While the fuzzer executes attacks, the sniffer (from the interaction layer) captures the traffic to be analysed by the verifier component (see next section). This component then displays and logs information when a bug is discovered, which would allow prompting the user to run other tests in face of the newly found problems.

B. Verification module

The verification module is composed by seven components, which are detailed bellow.

1) Media dissector: The media dissector is in charge of parsing the media layers within which DLMS packets are encapsulated. This component should supports the communication profiles described in the DLMS/COSEM specification.

2) DLMS dissector: The DLMS dissector receives preprocessed packets from the media dissector, parsing them in order to prepare DLMS packet fields and data to be inspected by the verifier component.

The best way to logically represent DLMS packets is in a tree diagram, as generic message types often contain more specific types, requiring branching out quite considerably. Therefore, ValiDLMS resorts to tree structures to keep information about a DLMS/COSEM packet organization and fields. For each packet, the dissector navigates in the tree and prepares the included data for inspection by the verifier component. Firstly, the dissector determines the type of the packet, and then notes down the specific context needed to interpret it correctly. Afterwards, at the leave nodes of the tree structure, it prepares the required and optional generic fields. At this stage no inspection is executed by the verifier. Fig. 4 shows a tree structure example for the navigation of a specific *get* request that asks for the time attribute of a server's clock.

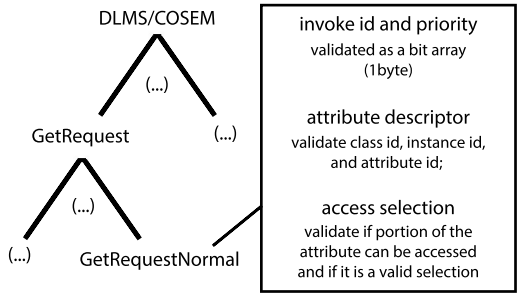

Fig. 4: DLMS packet with content structured as a tree, for an example get request packet validation.

3) Vulnerability database: This database holds information about vulnerabilities that were found in DLMS/COSEM [12]. Each database entry has a set of mandatory fields: a handle name for referencing; information on how it can be tested in the DLMS tree structure; a date of known disclosure; a risk assessment detailing how severe the vulnerability is, including a brief description of the possible impact on the vulnerable system; and resolution advice on how to remove the vulnerability or where to look for implementation mistakes.

As expected, this component needs to be regularly updated in order to be able to detect the latest vulnerabilities and help prevent their exploitation effectively.

4) Specification database: This database holds the information necessary to verify if the DLMS protocol is indeed well implemented by the sniffed packets, according to the DLMS/COSEM standard. Information about data format, message syntax, and correct values should be present. This specification is also structured as a tree. This tree can then be compared against the data processed by the DLMS dissector, and parsed along the respective nodes to perform additional validation at the leave nodes with the correct context.

5) Verifier: The verifier component is the most work-heavy component in the framework. It has two main verification tasks: (1) if the DLMS/COSEM messages are in compliance with the protocol specification, and (2) if the received packets carry malcrafted data attempting to exploit some vulnerability. For this purpose, the verifier uses complex pattern matching capabilities to check the contents in packet sequences, evaluate trees built from the validation data and verify them against captured and dissected packets. The verifier receives the context from the type of messages identified by the DLMS dissector component, and the captured message buffer, and then builds a general context for each connection.

The protocol verification is done using the tree built from the specification database. The verifier finds out whether or not existing fields should be carried in a packet, if the field data type is correct, and if field values are within those ranges deemed as acceptable in the specification. It also looks for the message patterns and the order of packet arrival. For example, it checks that requests precede responses, and that the security level is not downgraded in a sequence of messages. When looking for vulnerabilities, the verifier needs to parse the data from the vulnerability database. Then, it is compared to the arriving packets to determine if there is a match.

6) Display: The display provides feedback and online results to the user, employing colour schemes to facilitate the observation of found errors and vulnerabilities, and invalid messages, and distinguish between them quickly and easily. The information included in both database components is also utilized to provide comprehensive details about the findings. Colour schemes indicate the severity of the anomalies detected, showing by how much such anomalies put the network at risk or/and simply violate the DLSM/COSEM specification.

7) Logger: The logger produces readable and comprehensive logs from the running tests for offline inspection. The information presented in the log can be even more complete than the one presented by the display. Each log entry contains a timestamp, details about the problems and information that allows finding the offending packet or packets. The included details vary depending on whether a behaviour that violates the specification was detected, or a vulnerability was found.

V. Implementation

We have made a proof of concept implementation of the ValiDLMS framework, and deployed it in a testbed with components provided by an industry partner.

The DLMS/COSEM environment was set up with a network consisting of two data concentrators (DTC) and six smart meters communicating using PRIME PLC links. The DTCs acted as clients and were configured by the industry partner to carry out a number of predefined requests on the meters. They perform authentication with each meter using the LLS mechanism enables the client to authenticate by providing a password known to the server. However, no cryptography resources were in use in the tested configuration, as security was already offered and implemented by the PRIME communication layer.

In the interaction layer, a hardware based PLC sniffer was placed between the DTCs and the meters, capturing every packet in the network and storing them in PCAP format for further analysis by the tool.

The verification module was developed as an extension to Wireshark, taking advantage of its graphical interface and a few existing DLMS dissectors. The available dissectors had however to be enhanced, as we found some bugs that prevented the correct processing of the messages and they could not fully process the captured packets. Therefore, at first, we improved these dissectors, correcting their code in order to process any received messages.

We developed the verifier component, implementing functions to perform checks on message buffers and load information from the two database components. The current implementation performs a more thorough analysis of the *get* request messages, since they are the most prevalent. The specification database was built by extending the COSEM-pdu [17], a schema with all DLMS/COSEM message syntax with validation data. The vulnerability database was built using known DLMS/COSEM vulnerabilities [12].

```
184 nBox DLMS  60 Bytes: AARQ Logical_Name low_level (abcd1234)  cnf_blk: 0xffffff
184 nBox DLMS  60 Bytes: AARQ Logical_Name low_level (abcd1234)  cnf_blk: 0xffffff
167 nBox DLMS  43 Bytes: AARE Logical_Name accepted cnf_blk: 0x001014 max-pdu-size
167 nBox DLMS  43 Bytes: AARE Logical_Name accepted cnf_blk: 0x001014 max-pdu-size
137 nBox DLMS  13 Bytes: GET-request  Normal [Clock](time)
137 nBox DLMS  13 Bytes: GET-request  Normal [Clock](time)
142 nBox DLMS  18 Bytes: GET-response Normal    2017/09/12 03:41:52 local (7_DST)
142 nBox DLMS  18 Bytes: GET-response Normal    2017/09/12 03:41:52 local (7_DST)
155 nBox DLMS  31 Bytes: GET-request  Normal [Load profile with period 1 : Hourly
155 nBox DLMS  31 Bytes: GET-request  Normal [Load profile with period 1 : Hourly
```

Fig. 5: Packets marked in wireshark using colour schemes.

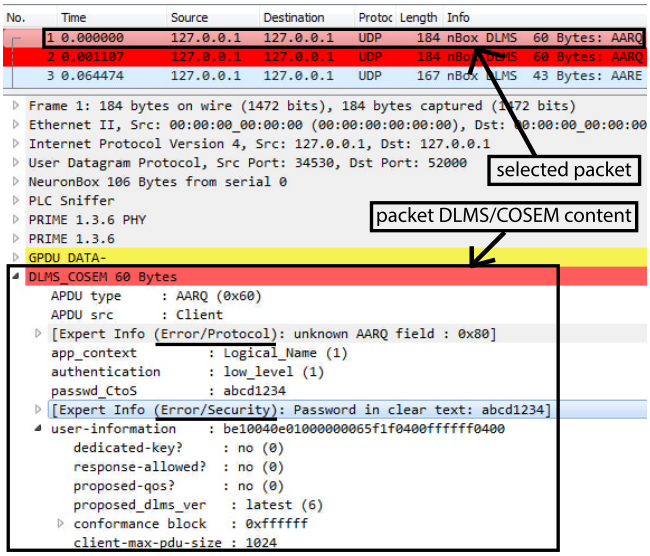

Fig. 6: Packet details with messages indicating problems found.

```
Packet 183: Malformed packet: unknown AARQ field : 0x80
Packet 183: Password in clear text in AARQ APDU. Password: abcd1234
Packet 184: Malformed packet: unknown AARQ field : 0x80
Packet 184: Password in clear text in AARQ APDU. Password: abcd1234
Packet 191: Malformed packet: check for packet segmentation or incompl
Packet 192: Malformed packet: check for packet segmentation or incompl
```

Fig. 7: Log entries produced by the tool.

The implementation of the fuzzing module was very limited, as we currently do not have access to a library that implements the PRIME communication layer. For this reason, implementing the fuzzer component and generating DLMS/COSEM traffic using our tool was beyond the bounds of possibility. We can still, however, capture and analyse traffic containing malcrafted data originating from attacks. Also, the attack database was populated with tests related to the exploitation of DLMS/COSEM vulnerabilities.

VI. Experimental Evaluation

The objective of the experimental evaluation was to answer the following questions: (1) can ValiDLMS be used to validate DLMS/COSEM implementations effectively? (Section VI-A); (2) can ValiDLMS be employed to find vulnerabilities in DLMS/COSEM implementations? (Section VI-B); and (3) can such a tool be utilized to test real industry based DLMS/-COSEM implementations? (Section VI-B)

The ValiDLMS tool was evaluated using a set of PCAP files containing traces captured from a DLMS/COSEM testbed with devices provided by our industrial partner. These devices and

DLMS/COSEM implementation are being nowadays considered for deployment across certain regions of the country.

A. Validation

The verification module of the tool was used to analyse the PCAP files. As the included packets were not encrypted, all DLMS message fields were visible as plaintext, and so the traffic could be read in its entirety. The tool was able to correctly determine the type and syntax of messages, and could perform a deep and complete analysis of AARQ messages and *get* requests and responses. The *get* request messages were validated and were shown to be correct by our tool.

On the other hand, the tool found small inconsistencies in the AARQ messages. These messages seem to contain a bug, as an additional non-conforming byte is being included in them. This problem occurs after one of the fields of the AARQ header. In addition, the tool also discovered bugs in the processing of segmented messages. Although apparently not very complicated, all these problems can have an important impact if devices from multiple vendors need to interoperate in the same smart grid deployment, as they may cause malfunctions under certain conditions.

While processing the PCAP file, the tool displays for quick visualisation the erratic packets using colouring rules (see Fig. 5). Red corresponds to the most severe problems, meaning that the error can put at risk the security of the smart grid infrastructure. Light blue indicates messages that are in conformance with the protocol specification. The fields of each packet can also be inspected in more detail, and problems are marked with colours. Once again, the colours are chosen depending on the severity of the errors. The tool also places informative labels near errors describing the problems found. Fig. 6 presents an example with the errors underlined in the package content.

The first question has positive answer, meaning that the implemented tool is capable of validating DLMS/COSEM implementations according to the protocol specification. Also, the tool had positive test results and was capable of withstanding incomplete or segmented packages, and tolerate and mark small one byte errors in packets.

B. Security auditing

Since the weak LLS authentication mechanism was employed and communications were transmitted in the clear, it was not possible to test very sophisticated security problems (e.g., information leakage in ciphered packets). In any case, the tool found problems in the AARQ messages, since the password could be observed directly. This would let any DLMS/COSEM client connect using it, and have authenticated access to the meters.

Fig. 6 displays the selected package, which is marked due to a password discovered in the authentication process. We can confirm such security leak by visualising the DLMS/-COSEM packet's content (bottom part of figure) and the label *Error/Security* near the error. For each error found, the tool logs information (and alerts) in a log file. Fig. 7 shows an excerpt from a log file produced by the tool when it finds security leaks and non-conformance problems.

The results show that the tool works in a smart grid environment and is able to validate and detect security flaws on an industry partner's DLMS/COSEM implementation, allowing us to answer positively to the second and third questions.

VII. Conclusion

The paper presents ValiDLMS, the first open-source framework for validating and securing DLMS/COSEM implementations in the growing smart grid industry, running over a power-line communication (PLC) profile, which is expected to be the norm for the industry in the recent future. The framework integrates a tool that was developed as an extension to Wireshark and was used to inspect an industry partner's DLMS/COSEM implementation. The results showed that ValiDLMS can effectively discover bugs and/or other non-conformance problems. We concluded that the framework can be applied by the industry to develop and integrate tools in order to test their own DLMS/COSEM solutions or those acquired from their business partners effectively.

Acknowledgements. This work was partially supported by the EC through projects FP7-607109 (SEGRID) and H2020-700692 (DiSIEM), and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/00408/2013 (LaSIGE).

References

[1] M. Carpenter, T. Goodspeed, B. Singletary, E. Skoudis, and J. Wright, "Advanced metering infrastructure attack methodology," 2009.

[2] DLMS User Association, *DLMS/COSEM Architecture and Protocols*, 2014.

[3] K. De Craemer and G. Deconinck, "Analysis of state-of-the-art smart metering communication standards," in *in Proc. of the 5th Young Researchers Symposium*, 2010.

[4] S. Feuerhahn, M. Zillgith, C. Wittwer, and C. Wietfeld, "Comparison of the communication protocols DLMS/COSEM, SML and IEC 61850 for smart metering applications," in *Proc. of the IEEE International Conference on Smart Grid Communications*, 2011, pp. 410–415.

[5] Netbeheer Nederland, *DSMR v4.0.4 P3 Companion Standard (Dutch Smart Meter Requirements)*, 2012.

[6] WireShark, "Wireshark," available at https://www.wireshark.org.

[7] CEN - CENELEC - ETSI Smart Grid Coordination Group, "Smart grid information security," CEN - CENELEC - ETSI, Tech. Rep., 2012.

[8] D. Podkuiko, "Vulnerabilities in advanced metering infrastructure," Master's thesis, The Pennsylvania State University, 2012.

[9] A. Faruqui, R. Hledik, S. Newell, and H. Pfeifenberger, "The power of 5 percent," *The Electricity Journal*, vol. 20, no. 8, pp. 68–77, 2007.

[10] DLMS User Association, *COSEM Interface Classes and OBIS Object Identification System*, 2014.

[11] M. J. Dworkin, *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC*, 2007.

[12] L. Weith, "DLMS/COSEM Protocol Security Evaluation," Master's thesis, Eindhoven University of Technology, 2014.

[13] M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerability Discovery*, 1st ed. Addison-Wesley, 2007.

[14] J. Antunes, N. F. Neves, M. Correia, P. Verissimo, and R. Neves, "Vulnerability removal with attack injection," *IEEE Transactions on Software Engineering*, vol. 36, no. 3, pp. 357–370, 2010.

[15] W. Jimenez, A. Mammar, and A. Cavalli, "Software vulnerabilities, prevention and detection methods: A review," in *in Proc. of the European Workshop on Security in Model Driven Architecture*, 2009, pp. 6–13.

[16] H. Dantas, "Vulnerability analysis of smart meters," Master's thesis, Delft University of Technology, 2014.

[17] DLMS User Association, "COSEMpdu XML Schema," 2015, available at http://www.dlms.com/COSEMpdu/.