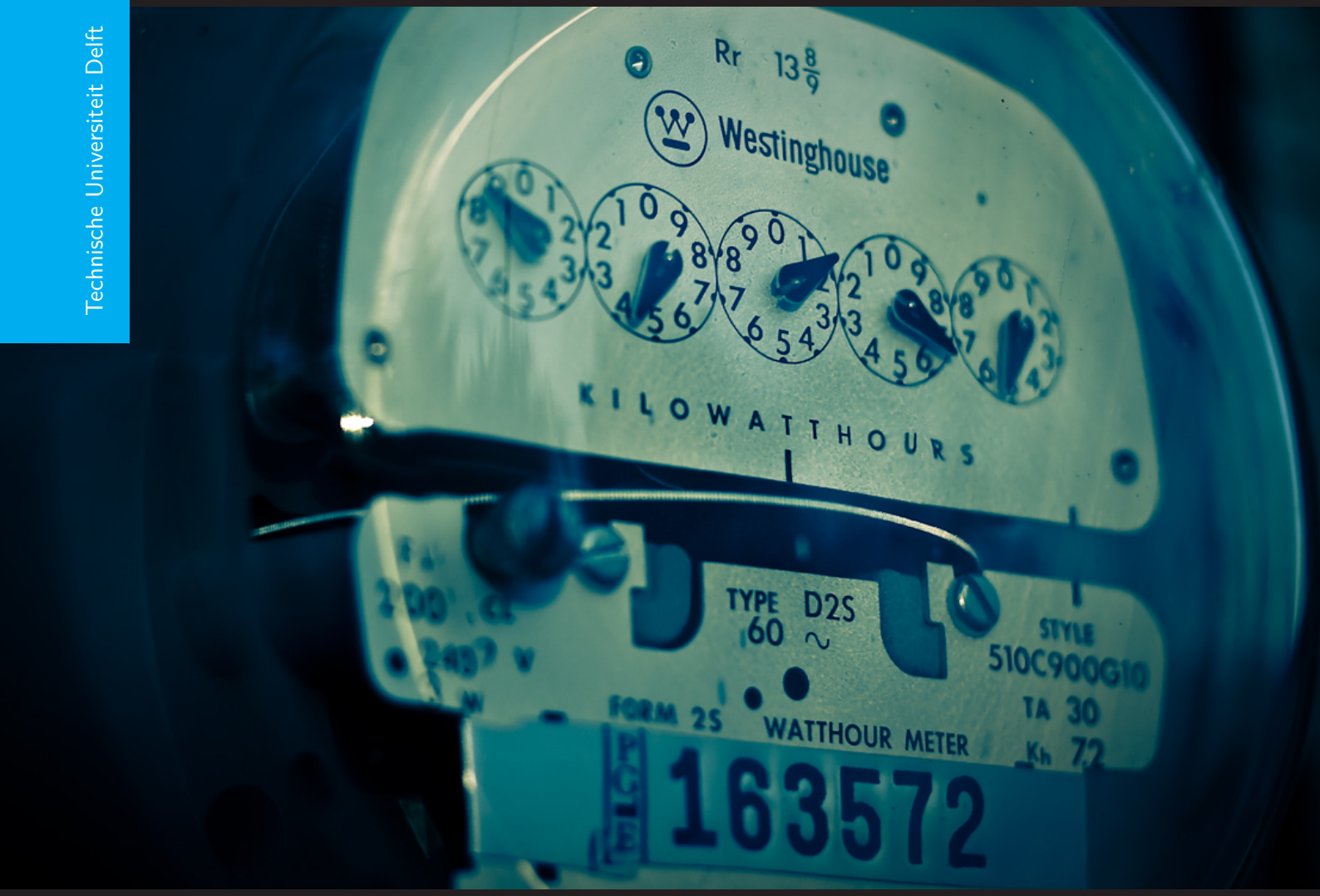
Vulnerability Analysis of Smart Meters

Henrique Dantas

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

Vulnerability Analysis of Smart Meters

by

Henrique Nuno de Matos Pais Neves Dantas

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Engineering

at the Delft University of Technology,
to be defended publicly on Thursday August 21, 2014 at 10:00 AM.

Supervisors:	Z. Erkin	
	C. Doerr	
Thesis committee:	J. C. A. van der Lubbe,	TU Delft
	I. Buhan,	Riscure BV

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

The online truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards.

Gene Spafford

ACKNOWLEDGMENTS

I am grateful for the continuous guidance and invaluable advice provided by Zeki Erkin and Christian Doerr, without whom this thesis would not have occurred.

I would also like to express my sincere gratitude to Riscure, in particular to Gerrit van der Bij and Ileana Buhan, for providing me with the opportunity, knowledge and resources to work on this research study.

My gratefulness is further extended to ENCS, specially to Raymond Hallie and Benessa Defend, who promptly made available all the facilities and assistance I required for this study.

Many thanks to all the friends I have made in Delft, who have filled my years in this small Dutch town with memories I will forever cherish.

My last, and most heartfelt words are to my parents and family, for their incommensurable support, and for providing me the means to follow my dreams.

Henrique Nuno de Matos Pais Neves Dantas
Delft, August 2014

Contents

List of Figures

List of Tables

1

INTRODUCTION

Critical infrastructures are, by definition, essential assets to the correct functioning of the economy and society. One of these infrastructures is the energy grid. Electricity is one of the most fundamental and visible commodities as any disruption in its supply can have profound effects on virtually everything else. Currently, the energy grid is undergoing significant transformation in the direction of smart grids that will enable a new range of applications, including but not limited to fine grained tariffs to decrease consumption fluxes, remote access to metering data and better management of micro-generation.

The smart grid effort has been jump-started by legislation, mainly in Europe and the United States, that aims to significantly transform the energy generation and distribution infrastructures. [6] estimates the current investment on the smart grid, also known as Advanced Metering Infrastructure (AMI), to approach $100 B, which has been accompanied by aggressive roll outs. An illustrative example is that of the European Parliament, which has mandated that by 2020, 80% of the households be equipped with next generation meters [7].

One of the marquee features of AMI is the ability to remotely monitor and control all grid activities. Smart meters are one of the major parts that make such capabilities possible. In addition, smart meters must allow the final customers to track, and hopefully reduce, their energy usage. [8] summarizes the major benefits of such devices in five points: (i) connectivity, (ii) continuous measurements, (iii) sufficient computationally capability, (iv) gateway functionality and (v) remote actuation.

The evolution in the energy sector has undoubtedly potential to improve the way society makes use of electricity however, it also opens the door to a new class of threats. As the current devices gain networking capabilities and the ability to execute remote commands they also become more exposed and more attractive to adversaries. Therefore, it is important to perform security audits of the different grid components.

Particularly, the networked controllable electricity meters, play an important role in assuring the security of the smart grid. Smart meters are situated in each household, hence are broadly available and can be easily probed by many people. As the physical protection is limited, it is safe to assume that curious customers will inspect the devices to try and understand how they work.

Furthermore, its communication capabilities would also enable an adversary of gaining control remotely. If a vulnerability is found that, for example, allows tampering with the reported usage, it can have important repercussions. The impact will be significant if the vulnerabilities are published online, thus endangering a large installation base of meters to be exploited. Since the smart meters are relatively new and there is little standardization with respect to security, many different vendors develop their own software, we envision that smart meter software is vulnerable to attacks.

Some of the plausible attack scenarios are as follows.

- *Disclosure of sensitive data* - Vulnerabilities in the smart meters that result in the leakage of consumption information can be used to ascertain if the home or shop owners are away.

- *Meter tampering* - Attackers may find desirable to inflate the utility bill of victims or unlawfully reduce their own.

- *Pivoting* - Penetration of meter defenses can be used for reconnaissance purposes or exploitation of other parts of the system. If the meter is connected to a domestic network, the opponent could use the meter to penetrate other devices in the house. Or, more creatively, be used to make free calls or grant Internet connectivity via the GSM connection that is common in smart meters. Alternatively it can reveal valuable information about sensitive parts of the smart grid network.

- *Remote command execution* - If an adversary gains complete access to a meter, it may be use it to extort individual victims, similarly to ramsonware. Meters could be switched

off, or internal security mechanisms be silently disabled so the electric installation or the meter itself would be damaged.

- *Grid destabilization* - Given the remote accessibility of smart meters, an attacker could gain controller of hundreds or thousands of identical metering devices and exploit vulnerabilities to simultaneously switch heavy loads such as electric cars on and off, with the intent to destabilize and bring down the power grid.

One technique that is widely used to find vulnerabilities in proprietary software is fuzzing. Very succinctly, it consists of trying numerous combinations of inputs, violating protocol or file format rules, to see how the target responds to unexpected requests. The simplest way to perform a fuzzing attack is manually. First one develops a crude version of a protocol client and then chooses a set of values the tester believes may cause software faults in the target system. However, this approach has various limitations: it is hard to fuzz complex protocols, reuse of code is not always possible, and the developers have to focus on more than the target protocol. To cope with such limitations, fuzzing frameworks have been developed, such as SPIKE [9], SNOOZE [2] and Peach [10].

These fuzzing frameworks typically target software applications running on traditional computing platforms, such as desktops. To monitor the working state of the targeted software, they might for example attach debuggers to the process or inspect memory consumption. Unfortunately, in embedded systems like smart meters, these capabilities are not always available and the fuzzing framework is usually physically separated from the investigated target. Therefore, to fuzz an embedded system like a smart meter, it is necessary to use a framework that supports the communication protocols available in the target and that allows specific types of monitoring developed for these equipments.

1.1. GOALS AND CONTRIBUTIONS

Currently, most of the literature [11–16] on smart grid security is primarily focused on architectural designs among other high-level approaches to improve the security of the AMI. In addition, various researchers and industry experts [17], have advocated guidelines and good practices that aim to raise awareness and assist smart grid stakeholders implementing this complex network in a relatively secure manner as this is a nascent field. In comparison, fewer tools to evaluate the security afforded by devices already in the market have been published. To address this void, this thesis presents a new fuzzer developed particularly for

DLMS/COSEM smart meters, called eFuzz.

It is important to note the focus of eFuzz is not to discover implementation bugs with security implications in a specific smart meter. Instead, eFuzz should provide a versatile and easily expandable tool to fuzz DLMS/COSEM smart meters, in order to complement human efforts. There are various contexts where eFuzz can be applied:

- A company hired to assess the security of a device would use eFuzz in an initial approach to autonomously detect possible points for further investigation. The respective report would serve as a guide to security experts and complement a full featured source code review.
- Original equipment manufacturer (OEM) can integrate eFuzz in their quality assurance efforts. Flaws detected at such an early stage are less costly to correct, and do not end up in production products where they may pose as a latent vulnerability.

For development purposes as well as to evaluate the functionality and effectiveness of eFuzz, a meter currently deployed in the Dutch market was used. In Chapter 5 the results of a vulnerability test, performed across various configurations, will be shown. The tests conducted on this specific smart meter are also compared with the traditional approach: an analysis made by a team of human experts.

The goals of this thesis are crystallized in three questions that served as guiding principles during the research efforts presented in the current document, which culminated in eFuzz.

Research Question 1. *Is it feasible to develop an automated tool with the goal of aiding security assessments of DLMS/COSEM meters, with sufficient flexibility to be easily adapted and expanded to different devices?*

Research Question 2. *Can such a tool be capable of uncovering vulnerabilities in said devices, within a time frame that is competitive with the conventional approach?*

Research Question 3. *In comparison with manual approaches, are the sets of vulnerabilities uncovered be each, largely overlapping or largely disjoint?*

These three are only a subset of possible questions that can arise during such a study. Regardless, they constitute the fundamental hypotheses addressed in the remaining chapters of this document.

1.2. THESIS OVERVIEW

This thesis is partitioned in six chapters, including the introductory.

Chapter 2 thoroughly presents the relevant related work. In addition to summarizing the state of the art in the three most relevant topics, namely smart grids, DLMS/COSEM and fuzzing, it introduces the reader to the same concepts.

Background knowledge is enclosed in Chapter 3. The contents of this Chapter, present in detail the DLMS/COSEM protocol, the various nuances of fuzzing and Peach. The reasoning for writing a complete Section on Peach is due to the fact that this framework forms the foundation of eFuzz. Therefore, having a thorough comprehension of its characteristics will immensely facilitate the understanding of the following chapter.

Chapter 4 is solely dedicated to eFuzz. It details how the Peach syntax is used to model the DLMS/COSEM protocol, as well as what modules were develop to compensate for the deficits of the framework.

Results and analysis are described in Chapter 5. In addition to the results obtained with a fuzzer, a quantitative and qualitative comparison with a manual security evaluation is shown.

Finally, concluding remarks, and potential avenues for future work are presented in Chapter 6.

2

LITERATURE REVIEW

2.1. SMART GRID

In Europe and the United States a significant engineering effort is taking place that will significantly transform the energy generation and distribution infrastructures. This investment, which according to [6] is estimated to cost around $100 B, has been spurred by laws that layout aggressive roll outs. For example, the European Parliament mandated that by 2020 80% of the households should be equipped with next generation meters [7].

The upgraded grid is designated as the *Advanced Metering Infrastructure* (AMI)[1]. The AMI should allow for monitoring and control of all grid activities, while ensuring the efficient bidirectional flow of energy and data connecting utilities and consumers, including all nodes in between. In addition, the Smart Meters must allow the final customers to track their energy usage, typically on the Internet or on a meter-hosted local server. The AMI infrastructure expands the capabilities of the simpler (and older) system, dubbed Automated Meter Reading (AMR), which simply collected meter readings and matched them with user accounts.

[8] summarizes the major benefits of Smart Meters over traditional metering equipments:

- Connectivity - smart meters can be remotely read, which implies they must be connected to a network for communication of data.

[1] On this document the terms Smart Grid and Advanced Metering Infrastructure will be used interchangeably.

- Continuous Measurements - enables the creation of high resolution energy consumption profiles.
- Complexity - devices must be capable to do verification of cryptographic signatures and encrypt their readings. Moreover the meters must have enough resources to accommodate communication modules that allow it to "speak" with other devices.
- Gateway functionality - should be able to deliver information to intelligent appliances or directly control them, according to the preferences of the users.
- Remote actuation - the control of the household appliances may be forwarded to authorized external entities.

Due to the described organization it is possible to operate the grid infrastructure remotely by sending commands to the appropriate network nodes, this is known as Distribution Automation (DA).

Although the particular technologies employed are meter-specific there are several common features that span all vendors. The data is collected locally and transmitted via LAN (Local Area Network) to a data concentrator. In turn, this equipment sends the information, processed or not, to the utility via a WAN (Wide Area Network) where it can be used for billing and other purposes. As previously mentioned, the opposite path is also possible and shall be used to remotely control various devices in the network.

There are two main methods to transmit data: Radio Frequency (RF) or Power Line carrier (PLC). The selection of the best option is necessarily tied to each scenario and should be determined by the utility, after careful study. Each has its *pros* and *cons* although analyzing these lays outside the scope of this document.

The authors of [18] claim that there is no single widely-accepted definition of Smart Grid and that reaching an agreement is a work in progress. However, they highlight three main Smart Grid goals that are consensual:

- Increased stability of the distribution network - aided by the ability of the grid operators to receive high quality measurements in real-time.
- Green operation - allows variables tariffs in accordance to time of day in order to mitigate demand fluctuations.
- Efficiency - remote maintenance and operation of grid endpoints allows faster responses with fewer personnel.

Although the Smart Grid concept is typically associated with electricity metering, it is also applicable to gas and water metering, apart from the ability to sell the commodity back to the infrastructure operator.

The Smart Grid is widely regarded as a welcomed (and necessary) upgrade that is advantageous to all parties involved. [4] does a good job of discriminating the main benefits for each stakeholder. Table 2.1 summarizes the results.

Stakeholder	**Benefits**
Utility Customers	Access to energy usage data More accurate billing Increased tariff options
Customer Service & Field Operations	Reduced cost of Meter readings Less need for hand-held reading equipments Decreased manual connections/disconnections
Revenue Services	Early detection of meter tempering and fraud Better billing estimates
Transmission and Distribution	Improved load management Better efficiency and reliability Facilitated medium and long term grid planning/design
Marketing & Load Forecasting	Detailed data on consumption usage & patterns
External stakeholders General	Diminished environmental impact Opens the door to value-added services on top of the Smart Grid infrastructure

Table 2.1: Main advantages of AMI to the involved stakeholders. Adapted from [4].

As previously mentioned, the AMI aggregates several components that need to work to-

gether in order to fulfill the *Smart Grid vision* and deliver the advantages outlined in table 2.1. One of these pieces can be the "Smart Meter gateway". [19] investigates the deployment of Smart Meters in Germany, where this component is required by law.

The main purpose of the gateway is to bridge the gap between the distribution and consumption devices. In other words, interface between the smart meter systems and the controllable household appliances, as well as the Internet connected remote systems (*e.g.* consumption display devices).

On the bright side, gateways can enable greatly improved consumer experiences, and consequently provide the utilities with new business cases. This is particularly important as it allows companies that are trading a commodity to offer unique, distinguishable services. On the negative side, they become responsible for vast amounts of sensitive information which may make consumers reluctant, at least in an initial phase. Furthermore, they constitute a desirable target for malicious attackers and therefore need to be resilient against a comprehensive class of attacks. To address these issues, the German Federal Office for Information Security has introduced a Common Criteria profile for the gateways with stringent security requirements.

The new functionalities enabled by the move to the Smart Grid help monitor, control, predict and plan energy usage and production. This is particularly important as various energy producers are added to the grid, notably with the integration of renewable energy sources whose production is dependent on weather conditions. Therefore, the amount of energy generated by traditional plants must be adapted with respect to aforementioned weather conditions and consumers' usage in a cost efficient manner. The increase in precision made possible by smart meters improves the quality of consumption predictions, and that helps drive the efficiency of the network upward.

The consumers also stand to gain from this transformation. According to [20] it is estimated that average consumption will drop by up to 15% when clients are given access to detailed monitoring. These savings will allow consumers to save money but also help combat global warming as the emissions produced by power plants diminish.

2.1.1. MOTIVATION FOR SMART GRIDS

Traditional electricity meters use dual tariffs (*i.e.* day and night) for billing purposes. The introduction of smart meters allows for much more refined tariffs, from 30 minute granularity up to real-time (in the range of few minutes) in the not so distant future. The improvements

in the resolution of measurements constitutes one of the cornerstone arguments for smart meters proponents.

From the point of view of the producer it leads to increased efficiency as there is close to real-time knowledge of the electricity consumption. In turn the customer can reduce costs, for example by instructing heavy load appliances to run when the electricity is cheaper[2]. This load-shifting capability significantly helps utilities as the peaks and valleys of the demand curves become less prominent.

In addition, by having access to more information the generation efficiency could be greatly improved as to better accommodate the actual (instead of the predicted) energy demands. Currently, consumption predictions are used to help plan medium and long term investments in the generation capacity. The short term variations are handled with the use of generators that are less efficient but more flexible. Therefore, it is easy to understand that the better the predictions, the more efficient the system. Smart meters have the potential to improve the quality of those predictions, even more since they can act as both sensors and actuators.

Notwithstanding the introduction of fine-grained tariffs, it is not realistic to assume that customers will be constantly looking at their meter to determine the best time to use their appliances. Thus, the process needs to be automated. In order to make this possible the meters must be equipped with data communication modules to orchestrate the different appliances, as to obtain the best €/kWh ratio without neglecting the time sensitiveness of each task. Obviously, the meter must also be able to query the utility for the present and future tariffs. Finally, the customer should also have access to a control panel to, at least, monitor electricity usage and set the preferred energy policies.

Smart grids shall allow for a better match between supply and demand. Being able to control demand to some extent, is also likely to improve the stability and thus the availability of the grid. Absorbing the large number of electric cars that are expected to enter the market, and are likely to be plugged in at night is also facilitated by this change. A more subtle effect is enabling better protection against cyber or offline attacks on the energy grid, as the new found flexibility can help prevent cascading failures and prevent ripple effects. In addition, this system will allow the infrastructure to better cope with micro-generation from alterna-

[2] Currently it is usual to find day and night tariffs in most countries. Therefore consumers can run some appliances, *e.g.* washing machine, at night. However by having more price points, time-sensitive appliances can also take advantage of the cost fluctuation. As an example, an air conditioning system may run in power-saving mode for 30 minutes and then return to normal after that period has passed, and the cost has gone down.

tive (fluctuating) sources. For example, if there is a sudden increase of solar power injected in the grid, the utility may decide to lower the sell rate thus enabling consumers to antedate the use of certain devices that would otherwise run at a later time. Ideally, it would be indistinguishable if the renewable power originates from the major utility-controlled photovoltaic panels or from individual micro-producers.

2.1.2. MAJOR CHALLENGES

The most pressing question regarding the AMI model described in the previous sections relates to privacy. One of the relevant questions is, should all measurements be sent to the utility or should it only have access to sampled or aggregated data? In the former case, it maybe possible to infer the user's lifestyle and routine from the data, information with a high value of monetization. Although these arguments are fascinating topics for discussion and require the scrutiny of public debate before the *Smart Grid vision* becomes a reality, they do not necessarily pose technical challenges, and are thus outside the scope of this thesis.

Other than the security issues already mentioned the heterogeneity of the smart meter environment is bound to raise many complications, at least initially. There are several competing standards aimed at this market with numerous implementations available. This uncertainty in combination with the vast number of smart meter vendors results in a complex infrastructure that needs to function correctly while still being secure.

In order to protect meters from potential attacks is important to understand which attack vectors these systems are exposed to and categorize the type of entities that may try to exploit them.

[20] outlines several attacks that smart metering systems can be subject to.

1. Attacks on communication module - the first option is to tackle the communication module of the smart meter. It can be accomplished through physical access (typically it is only protected by a seal) or through the communication network.

2. Attacks on utility server - the server must be available from any point in the network and is thus exposed to attacks.

3. Attacks on service provider - instead of targeting the server directly the attacker may try to infiltrate the utility network. This can be accomplished via a malicious email attachment, infected USB drive, etc. This type of attack is similar to the one used by the infamous Stuxnet malware that was able to penetrate the air gapped network of Iranian

nuclear facilities.

4. Attacks on the communication network - another possibility is to infiltrate one of the communication networks between the meters and the utility servers. Moreover a man-in-the-middle type of attack is also possible in this scenario.

Moreover, the paper identifies three categories of attackers:

1. Outsiders - includes criminals, activist, terrorists or foreign states. These can be powerful and resourceful adversaries that will likely go to great lengths to accomplish their objectives, which can vary from provoking chaos to covert reconnaissance.

2. Homeowners - expected to be the origin of most attacks, similar to what happened in the pay TV industry. Typical attacks aim to reduce the monthly bill and are often performed by people with limited knowledge. However, if an exploit is made public (*e.g.* through the Internet) these attacks can have a significant impact in the utility's ability to collect the correct fares.

3. Energy provider employees - disgruntled or former employees constitute one of the more serious threats, as they have intimate knowledge of the infrastructure and may even possess access permissions that can be leveraged to create significant harm. Their motives can be born out of frustration with their employer or financial interests.

Some of the greatest risks to the successful implementation of the Smart Grid are related to the vast amounts of private data that must be transmitted and the potential for meter tampering and fraud. In particular, if a security vulnerability is found on a class of meters that leads to widespread fraud (*i.e.* artificially lowering the usage or not reporting measurements) it can have devastating consequences for the utilities.

Privacy concerns abound, nonetheless they and must be handled properly as the amount of sensitive personal information that can potentially be disclosed is significant. According to [6], equipment vendors claim that the move towards smart grids has provoked excessive regulation that has increased the costs for little benefit. Additionally, the existence of a remote off switch in all electricity meters makes them a desirable target for attackers, since a successful denial-of-service attack could cause great disruption. Perhaps more importantly it might create several issues which utilities have no experience addressing. The complexity of the protocols involved and the new instructions, APIs, applets, etc. supported by the me-

ters make it likely that possible exploitable bugs arise which require speedy deployment of firmware updates.

In order to mitigate some of those risks [8] recommends some measures. First, aggregation of data according to its purpose (*e.g.* billing) or anonymization techniques. Second, deployment of homomorphic encryption (*i.e.* performing calculations on encrypted data) to prevent the need to decrypt or transmit plain-text information. Third, updated legislation. Although the previous approaches are of technological nature, regulatory approaches, such as financial penalties, are also worth investigating. The downside of the latter is that it can only be applied retrospectively, while the previous can prevent the incidents from occurring in the first place.

Moreover, there are severe conflicts of interest between the various Smart Grid stakeholders. While in one hand the governments want to reduce energy usage, the utilities often rely on confusion pricing to maximize sales and profits. Also the availability of vast hordes of consumer data to utilities can adversely affect competition via increased lock-in.

Due to the sensitivity and potential monetization of consumer data it is crucial to enforce access control rules. However it is first necessary to determine who owns the data: the utility since they own the meters or the customer. [6] claims that due to pressure from privacy groups some EU countries are giving ownership of the data to the customers and prohibiting the utility from sharing any data without consent.

In [18] researchers from C4Security take a practical approach and attempt to demonstrate the need to secure the smart grid by exposing nine critical attack vectors and vulnerabilities. Their main goals are to raise awareness for these topics and instigate a discussion about potential remedies.

In a Smart grid the distribution level can remotely read data from the meters and send connect and disconnect commands that affect the supply of customers. Therefore it can be viewed as a micro-SCADA command and control system. One of the biggest security challenges with the Smart Grid data network is the fact that the various nodes are not physically protected. They are located at homes and businesses where there is no possibility to restrict access or quickly detect tampering. Thus it is reasonable to assume they will be inspected and tampered with by interested and curious customers.

Since each meter is a node in the Smart Grid network the management applications and services are exposed and available for all nodes. Therefore an attacker can cause network-wide changes if no explicit constraints are in place.

The authors were able to encounter several meters that did not possess any authentication or encryption support which makes them vulnerable to an attacker impersonating the control center and sending unauthorized commands to other meters. Moreover although some of the protocols support encryption these features were routinely found to be disabled, as they also support no authentication or no encryption modes.

Another issue identified by C4 was the potential for a man-in-the-middle attack that could lead to artificially lowering the reported usage. The protocol between the master meter and the slave is considered to be less important as its impact is restricted to one household.

The protocol implementations were found to be vulnerable to Buffer Overrun/Overflow attacks through malformed requests as they made certain assumptions about the received data that may be exploited. These vulnerabilities can be used to interfere with system stability, change set parameters or even execute arbitrary code.

A final attack vector highlighted in the paper is firmware upgrades. Their existence is important to fix vulnerabilities however if not properly secure can be abused by attackers to deploy malicious versions. Thereafter the meter would be at the will of the attacker and most likely could only be restored to the original state after being shipped back to the manufacturer.

[21] also investigates the security of Smart Metering, albeit focused on the Software Development Life-cycle.

The authors use a Threat Modeling process at the design phase of software development to support the methodological creation of a trustworthy system design and architecture. Thereafter the process is applied to a Smart Metering Gateway.

The gateways are used for prosumers (*i.e.* simultaneously producers and consumers) to communicate with their consuming and generating devices as well as the Smart grid operators. They identify the gateway as representing the Smart Grid's "security core" therefore having great influence in the actual and the perceived security of the infrastructure.

The main goals of Threat Modeling, as defined by the Common Criteria profile are identifying potential attack vectors and associated risks and vulnerabilities. In addition to this systematic process the authors suggest that extra steps should be considered before deployment of a product:

- Static Source Code Analysis
- Penetration Testing

- Dynamic Analysis and Fuzzing
- Exploratory testing and manual code auditing
- Back door Detection

[22] takes yet another approach to improve the security of the Smart Grid. The researchers detail a framework to help mitigate the issues associated with privacy that arise with AMI.

The proposed frameworks allows Smart Grid stakeholders such as consumers, utilities, and third parties to have access to spatially and temporal aggregated data. Thus protecting the individual customers from being directly linked to specific readings.

The authors introduce a set of new functional nodes, namely the "Privacy Preserving Nodes" (PPNs). The PPNs collect sensitive data masked by a shared schemed featuring homomorphic properties. Thereafter they aggregate it in the masked domain according the the consumer's needs and access permissions. These can recover the original data by collecting multiple shares from the PPNs.

The framework uses an Integer Linear Programming formulation and a greedy algorithm to transmit information between the different entities. Moreover a study of the scalability of this solution is also performed, under the assumption of error-free and error-prone networks.

As previously explained, the introduction of Smart Grid will enable new added-value services to improve the customer experience. Therefore it is safe to assume new actors will play a role in the management and creation of these services and the supporting infrastructure. These parties include both the service providing enterprises and the regulatory authorities.

To accommodate for this scenario the framework must possess a service platform ready to support new applications provided by traditional utility companies and third parties that aim to offer new valuable services. The main difference with relation to the traditional system is that both the resource itself (*e.g.* electricity) and the information of its use and production are assets with economic value.

Taking the previous into account the authors recognize the following contributions:

- Design an high-level architecture for a privacy conscious infrastructure. It add new functional components to the smart grid, namely the Privacy Preserving Nodes (PPNs). In essence the PPNs collect and aggregate customer data masked by homomorphic encryption to protect the final user's privacy.
- Identification of design problems related to information flows and the PPNs.

- Modeling of the aforementioned issues by means of an Integer Linear Programming formulation. This model is NP-hard, thus a scalable greedy algorithm is also presented.

- Evaluation of the scalability of the infrastructure. This is performed for reliable and non-reliable communication networks.

The different measurements belonging to a producer are divided in multiple shares using a homomorphic secret sharing scheme. Each of the individual shares is sent to different PPNs. Assuming t shares exist, then to obtain individual measurements at least t PPNs would need to collude.

Nonetheless they are still able to concurrently operate on the shares originating from different producers or at different times to determine the required information. In turn, the consumer receives all shares and can run recovery algorithms to gain access to the complete picture.

By virtue of the homomorphic properties of the scheme, the consumer obtains the aggregated results (but no information relative to the individual measurements) by running the recovery algorithm over the sum of shares.

The authors studied the scalability of the proposed framework and concluded that in an error-free communication network the architecture can incorporate millions of smart meters. On the other hand, when the communication medium is not ideal the number of necessary PPNs increases rapidly which limits the overall scalability of this system.

[23] analyzes the privacy-energy efficiency when energy harvesting (*e.g.* photovoltaic panels) and storage units (*e.g.* rechargeable batteries) are present. The authors quantify the privacy of a user's energy profile in terms of the *information leakage rate*. This metric is defined in the paper as "the mutual information rate between the real energy consumption of the appliances and the smart meter readings".

The methods and models used in this paper are purely theoretical, in particular from an information theory perspective. Therefore, in the context of this document, there is little interest to delve into their details. Notwithstanding the main findings of the paper provide interesting insights.

The authors conclude the information leakage rate can be significantly reduced when both energy harvesting and storage units are present. In particular, energy harvesting increases privacy by diversifying the energy source. While the storage unit is able to hide the load signature from the utility provider. As a downside these units also lead to wasted energy.

To determine the best trade-off between energy efficiency and information leakage, the researchers fixed the energy harvesting rate and numerically obtained the curve between the information leakage and wasted energy rates. Using these results utility providers and other interested parties are better equipped to find an adequate balance between acquiring detailed metrology data and ensuring an acceptable level of privacy for their consumers.

[24] considers the large scale deployment of smart meters containing a remote off switch constitutes an important new cyber-vulnerability. The main purpose of such a switch is to act in situations when clients default on their payments, by redirect them to prepaid tariffs, supporting interruptible tariffs or to assist in crisis situations when energy is in short supply and should be directed to crucial infrastructure (*e.g.* hospitals).

This attack vector is of great importance and creates challenges that utilities have not yet faced. From the perspective of an attacker (*e.g.* foreign state, terrorist organization, *hacktivist* group) the value of being able to severely disrupt the power supply cannot be overstated, since soon after electricity stops so does everything else. Thus far this could only be achieved through an attack on critical generation, transmission and distribution assets, which are easier to defend.

The main contribution of the paper is a metering architecture that aims to institute trusted parties in the system, in a similar fashion that is used for the Internet, *i.e.* using a Public Key Infrastructure (PKI).

Communications between the meter and the head-end will be protected using cryptography. The details of the protocols for the different metrology equipments are still a work in progress.

A possibility for a PKI system would be for the private key of the regulator to be "hard-coded" in the meter as trusted, and then use such key to sign and authenticate the energy companies' keys. This way adding new utilities or revoking compromised keys would be simple.

An alternative method would be for a set number of established energy providers to sign the key of a new entrant. However, there is no benefit for the establishment to do so and therefore the process would likely be slow and frustrating.

Yet another option would be for the meter vendors to embed their keys in the meters and thereafter sign the keys of the utility companies operating in the country of sale. This scheme is able to solve several issues related to authentication and trust in the system. However it raises others, such as accountability and liability. Would vendors be willing to take the nec-

essary steps, and associated cost, to ensure the security of their private keys and to act swiftly in case of breaches? This is particularly important as these players have little experience in the area of PKI's.

The authors conclude the paper urging for action, and reiterating that these issues should be discussed prior to large-scale roll-outs.

2.2. COMMUNICATION PROTOCOL

This section will focus on introducing and analyzing the DLMS/COSEM protocol used by various Smart Meters, based on published literature. In addition competing standards will also be used to assist in this study. Further details on the DLMS/COSEM protocol are shown in Section 3.1.

2.2.1. INTRODUCTION TO THE PROTOCOL

DLMS stands for *Device Language Message Specification* and consists of a general concept for abstract modeling of communication entities. On the other hand, COSEM is derived from the *COmpanion Specification for Energy Metering*. It includes a set of standards that defines the rules for data exchange between devices, such as an energy meter and a data accumulator.

[25] is the official website of the DLMS User association and provides a simple introduction to the protocol. Registered members may also obtain the complete DLMS/COSEM specification, also known as the colored books, namely *Blue book*, *Green book*, *Yellow book* and *White book*.

Not surprisingly each describes a part of the set of standards. The Blue book focuses on the COSEM meter object model and the OBIS (OBject Identification System) codes. Next, the Green book describes the Architecture and Protocols. Then the Yellow book pertains to conformance testing. The DLMS User Association is responsible for issuing the DLMS/COSEM conformance certification. And lastly the White book is dedicated to the glossary of terms.

Together they provide several features:

- An object model to view and access the different functionalities of a meter.
- An identification system for all data.
- A method for communicating with the model.
- A transport layer to accommodate the information flows between the meter and other devices.

The protocol is based on the OSI (Open System Interconnection) seven layer model. However they are collapsed in four: physical, data link, transport and application layers.

Prior to exchanging metering information an association must be set up, initiated by the client, through the object model interface. From that moment the server is also able to send notifications without explicit request.

Clock synchronization and transmission of measurement profiles are also features of DLMS/-COSEM. Finally, it includes authentication and confidentiality services based on symmetric key encryption.

Smart meters are fundamental for the success of AMI. The addition of new features to these devices poses many unique challenges for the utility providers and for the manufacturers. Adjusting current and future software and hardware to enable these services, ensuring compatibility and interoperability are just some of those. DLMS/COSEM aims to help solve these complications and facilitate the process of smart meter development.

Section 3.1 provides a more detailed study of the specifics of the DLMS/COSEM protocol.

2.2.2. COMPARISON WITH COMPETING ALTERNATIVES

According to [26], at the current moment DLMS/COSEM is the best option to address the aforementioned difficulties and make meters smart and able to talk with each other. Despite this conviction and the popularity of the standard, it is advantageous to compare it with similar solutions, if only to understand what alternative approaches can be adapted or be used as guidance to solve current DLMS/COSEM limitations.

[27] reviews DLMS/COSEM and IEC 60870-5, which the authors consider the most widely accepted standards for smart grid industries in Europe and other parts of the World. The authors aims to identify the main goals of each and demonstrate why they successfully accomplish them.

DLMS/COSEM is intended for interaction between revenue meters (including electric, water, gas, etc.) and a remote center. On the other hand IEC 60870-5 targets systems used for supervisory control and data acquisition (SCADA) and monitoring of power system automation applications. Both standards (or set of standards) are particularly tailored for the equipment market and thus can potentially be used in the Smart Grid.

DLMS/COSEM supports a wide array of communication methods such as Ethernet, GSM/-GPRS, WiFi, ZigBee and PLC. From the perspective of buyers it allows for great flexibility, however from vendors it increases the production cost and makes the certification process more

burdensome.

DLMS/COSEM is composed by thee logical layers:

1. COSEM layer - provides an object model that implements the necessary metering functions.

2. DLMS Layer - this layer provides access to the COSEM objects. It is responsible for the assembly and transport of messages.

3. Communication protocol layer - transforms the DLMS messages to the desired communication protocol based on the lower layers.

IEC 60870-5 includes three layers of the OSI model: application, link and physical layer. This suite enables two different transmission procedures, unbalanced and balanced.

For unbalanced operations, the controlling station (typically a SCADA system) controls the data by inquiring all the controlled substations. This constitutes a slave-master model, where the slaves can only reply to explicit requests by the SCADA system, the master.

Alternatively, in the balanced procedure each station can initiate a dialog. Stations can act simultaneously as controlling or controlled stations.

The IEC 60870-5 application layer incorporates several basic functions. These include data acquisition, event acquisition, clock synchronization, command transmission, among others.

It expected that most often the meter (server) will assume a passive role and only respond to requests from the utility controlled devices (client). However, it is sometimes necessary to report unexpected or sudden situations hence the ability to trigger event notifications. As previously stated, this feature is supported by the DLMS/COSEM suite.

Vendors wishing to adapt their meters to the standard can leverage existing efforts from specialized businesses. Open-source projects backed by Gurux [28] and jDLMS [29] are available and can be readily integrated. However, they are often incomplete and require additional development. In contrast, proprietary solutions are mature and reliable. For example, Kalkitech [30] is the most popular independent provider of such protocol stacks. Nonetheless some vendors, such as Texas Instruments [31], develop libraries compatible with their equipment and make it available to clients.

[1] presents four application layer protocols in the AMI context: DLMS/COSEM, the Smart Message Language (SML), the mappings to MMS and SOAP from IEC 61850. The emphasis of

the paper is placed on the protocols' use over TCP/IP, thus suited for Internet communication or for local traffic via Ethernet or WiFi.

Figure 2.1 illustrates a generic communication topology between metrology devices and the utility provider. The application layer protocols described in the paper can be used for TCP/IP traffic and therefore are applicable for Internet communication on (c) and (e) and over local (*i.e.* within individual households) networks such as Ethernet and WiFi on (a).

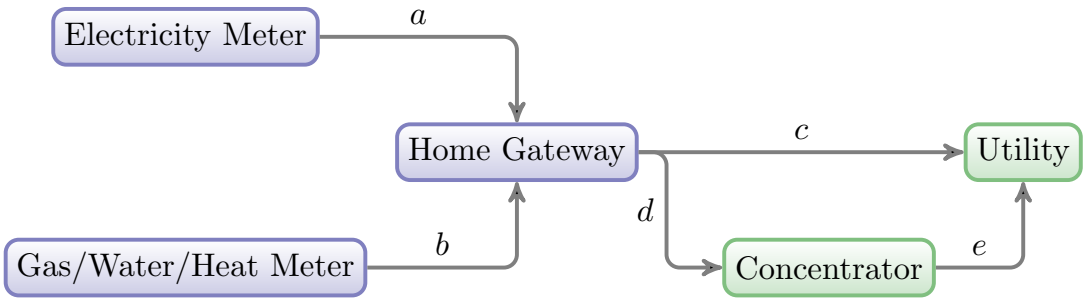


Figure 2.1: Communication Topology diagram, adapted from [1].

The Smart Message Language (SML) is particularly successful in Germany but is rarely used in other countries. It was developed as part of the Synchronous Modular Meter (SyM2) project. SML distinguishes itself from IEC 61850 and DLMS/COSEM in the sense that it defines messages instead of an interface object model and services to access it. There are two types of messages, requests and responses, however unsolicited responses also exist. These messages enable the communication of load profiles and associated digital signatures. The standard also supports firmware updates and clock synchronization. Encryption, on the other hand, is not directly available for SML, but SSL/TLS can be used to secure TCP/IP traffic.

IEC 61850 was originally designed for use in the automation of electrical substations but has been extended for hydroelectric power plants, wind turbines, and other distributed energy resources. It uses the same client-server principle as DLMS/COSEM. The server provides an object model interface accessible through standardized services. The realization of these services depends on the specific communication mapping used, *e.g.* MMS (Manufacturing Message Specification) or SOAP (Simple Object Access Protocol).

From the three, SML is the most unique. Since it does not rely on a model interface it does not standardize the functional capabilities. This allows for greater flexibility but in order to guarantee interoperability other standardization bodies must specify more details.

One of DLMS/COSEM advantages is that it is already popular, in particular in Europe, and

therefore numerous parties are working together to add missing features to the specification. A current limitation is that DLMS/COSEM requires symmetric key encryption which may be a significant hindrance in various scenarios.

Finally, IEC 61850's particular strength is that it may also be employed to control smart grid applications, other than smart meters.

The authors conclude by stating all standards have their advantages and disadvantages but no one is superior in all regards. With respect to encoding, DLMS/COSEM and SML stand out as the best solutions. A similar study with protocols such as ZigBee Smart Energy 2.0 and ANSI C12.19 is identified as promising future work.

2.3. FUZZING

This section briefly introduces the concept of fuzzing and is mostly dedicated to the state of the art in the field. However, due to the importance of the topic in this thesis, there an additional section, Section 3.2, which provides more information.

2.3.1. DISCOVERING SOFTWARE VULNERABILITIES

Software vulnerabilities are the main culprit for technology security incidents. These occur with a worrying frequency and reports of such events are a recurring occurrence in media, including mainstream newspapers and television channels. Therefore there is significant interest originating from all corners of society on how to detect and fix software vulnerabilities as quickly as possible.

[32] discusses three software vulnerability techniques, namely static analysis, fuzzing and penetration testing.

Static analysis inspects the structure of the application in search for potential issues. Therefore it requires access to the source code, although it does not involve program execution. Other drawbacks of this technique are the high number of false positives, no design bugs detection and the need for human verification.

Fuzzing, on the other hand, is a highly automated testing technique. It is analogous to brute-force testing as it involves using numerous inputs to provoke the target to crash. The tests generated by the fuzzer can be based on permutations of valid inputs or created from scratch based on a model of the target. Traditionally fuzzers do not expect access to the source code of the target applications. Disadvantages of fuzzing include high randomness which may require long running times to find bugs.

Finally, penetration testing involves simulated attacks on a system to assess its resilience. There are several types of penetration testing depending on how much access to the system the tester is awarded. This process is typically performed by an external entity, that specializes in security audits. Two problems with penetration testing are the thoroughness of the test is completely dependent on the skills and experience of the involved professionals and second, it exposes the system to an external party which may be dangerous.

[33] presents a new approach to guiding penetration testing of several devices deployed into the Advanced Metering Infrastructure (AMI) to evaluate their security. They recognize that current industrial penetration testing strategies have not been sufficiently adapted to address the security concerns of the AMI stakeholders. For example the utilities' concerns regarding fraud and denial of service. In particular they show that focused penetration testing identified various security issues with two popular AMI vendors, but can be applicable to many more vendors.

In summary all three discussed options have advantages and disadvantages. To improve the security of a system it is important to understand these and, most likely, employ a combination of them.

In this thesis we will focus solely on fuzzing for three prime motives:

1. We do not assume access to the source code implementation of the DLMS/COSEM protocol.

2. Fuzzing is particularly adequate for finding vulnerabilities in network protocols.

3. Since the protocol is somewhat recent it is expected that different vendors will have slightly contrasting implementations. With fuzzing it is relatively inexpensive to accommodate these variations thus increasing the applicability of such a fuzzer.

2.3.2. INTRODUCTION TO FUZZING

Fuzzing originated in the University of Wisconsin in the context of a 1988 class project [34]. The goal was to develop a command-line application to study the robustness of several UNIX applications. In essence this program, the original fuzzer, simply generated a long stream of random characters and used it as input to the programs under test. This work resulted in the "Empirical Study of the Reliability of UNIX Utilities" [35] where the authors present and analyze the results of applying this novel technique. It is interesting to note that even in this preliminary stage, the fuzzer was able to crash 25 to 33% of the target programs.

Over the years fuzzing developed significantly and we can now consider various types of fuzzers depending on the type of applications they target.

Network fuzzing is applied to network applications, typically packet based. This category includes testing software that is not running on the local machine, or requires a network protocol. It is important to take some care in formulating input data for protocols otherwise the packets will likely be dropped without further processing. In order to increase the efficiency of such an attack, size fields or hashed values for example, must be correctly computed.

File fuzzers audit software that reads and writes data to files. The idea behind these fuzzers is similar to the previous but instead of sending packets through a network, a file fuzzer crafts local files and monitors the target application as it tries to open them.

When comparing fuzzers the crucial metric is efficiency. The basic idea behind this technique is simple, however to be able to write fuzzers that discover vulnerabilities in a reasonable amount of time, enough intelligence must exist, either included in the tool, transmitted by the auditor, or most often, a concoction of both. The next paragraphs showcase different fuzzing techniques and tools that propose solutions for this problem using novel approaches.

The remainder of this Section focuses on the state of the art in fuzzing. In addition, Section 3.2 provides more information on what fuzzing is and what it entails.

2.3.3. FUZZING FRAMEWORKS

The traditional way to perform black-box tests of network protocols is quite straightforward. First one develops a crude version of a protocol client, and then manually picks values one believes may cause software faults in the target system. Typical attempts include using incorrect types (*e.g.* if the protocols expects an integer, send a floating point number instead), or using negative values when only positive are expected, among others.

This first, purely manual, approach may prove successful but it possesses various inefficiencies:

- If the protocol is accessible through API's or the source code is available, the tester is likely to make the same assumptions as the original developers, thus reducing the coverage of the tests.

- Developing a client for a complex protocol involves a great effort which significantly increases the cost of testing. In addition, it is probable that such a client can not be easily ported to other protocols or architectures, reinforcing the cost argument.

- Often testers have limited knowledge of the protocol they are attacking.

To address these limitations Dave Aitel proposed a block-based fuzzing framework designated SPIKE [9]. The framework was developed to simulate network protocol clients and automate black-box testing. It is implemented as a C-like API and scripting language which is used to leveraged the tester's knowledge of the protocol.

SPIKE is able to isolate lower level protocols from the higher levels. This allows the calculation of size fields without constructing all higher layer protocols. In other words, the process of determining the final size is deferred until the blocks are closed.

Another task where SPIKE can greatly reduced development time is dealing with different representations of data types. To accomplish this the framework provides a collection of encoding routines that can be used to accommodate the protocol's specification.

According to the author, by using the SPIKE framework he was able to uncover dozens of new vulnerabilities in different protocols of varying complexity. The paper claims that black-box testing continues to be the major source of exploitable real-world vulnerabilities and thus tools such as SPIKE can play an important role in auditing efforts. In addition, the flexibility and relative simplicity of SPIKE allows software vendors to quickly and efficiently test their solutions before deployment.

SNOOZE [2] is another network protocol fuzzer that implements a stateful fuzzing approach. Therefore a tester can describe the stateful operation of the protocol and specify the messages to be generated in each state. Moreover, it provides fuzzing primitives that are specific to certain attacks and thus allow the user to direct her efforts solely on a class of vulnerabilities. The authors use the Session Initiation Protocol (SIP), used for VoIP, to demonstrate the aforementioned properties of SNOOZE.

SNOOZE has six high-level components: *Fault Injector*, the *Protocol Specification Parser*, the *Interpreter*, the *Traffic Generator*, the *State Machine Engine*, and the *Monitor*. Figures 2.2 and 2.3 illustrate the architecture of SNOOZE.

The Fault Injector alters fields of valid messages to trigger faults on the target system. It relies on values that typically cause issues to reduce the state space.

The Parser, not surprisingly, parses the protocol specification so it can be used by the remaining SNOOZE components.

The Interpreter runs the fuzzing tests. It should receive three inputs. First, a set of protocol specifications that describe the protocol, *e.g.* the format of the headers. This file should be written using the Extensible Markup Language (XML). Second, a set of user-defined fuzzing

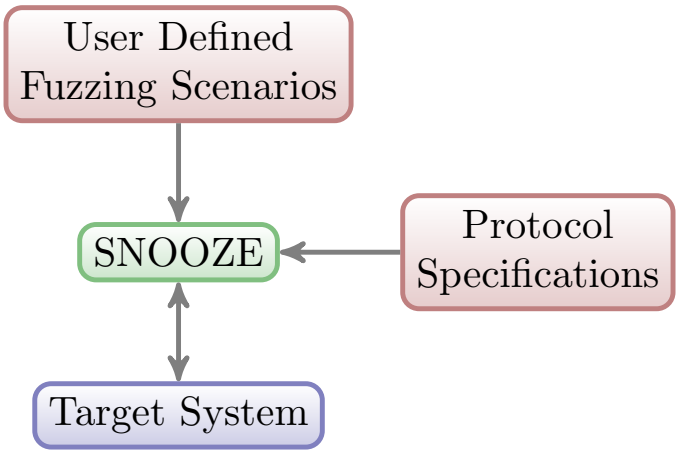


Figure 2.2: High-level Architecture of SNOOZE. Adapted from [2].

scenarios, that are written as Python scripts. They are responsible for assembling the messages appropriate for the target based on the other components of SNOOZE. In practice it means sending packets with some of the fields fuzzed. And finally a module implementing scenario primitives. These are basic operations invoked by the user that the system uses to derive test cases.

Next, the Traffic Generator assembles the packets prior to sending them to the target system. It transforms the messages created by the aforementioned user scenarios into network packets, while taking care to update their checksums, and other variable fields.

The State Machine engine, as implied by the name, keeps track of the state of the protocol by logging the received and transmitted messages.

Finally the monitor keeps an eye on the target system to erroneous behavior, such as segmentation faults, unexpected outputs, etc.

[5, 36] propose a theoretical approach for fuzzing the most traditional targets: large-size applications, with proprietary source code, and with limited knowledge of the input specifications (obtained for example through reverse engineering). In particular, they focus on improving fuzzing through taint analysis and increased coverage. This approach is suitable for both file and protocol fuzzers.

The researchers' method starts with analysis of the binary code. This step consists of identifying potentially unsafe assembly sequences by comparing them with known vulnerability patterns. Thereafter the target is executed and analyzed using taint analysis.

The main principle behind this technique is first to mark untrusted data as tainted and

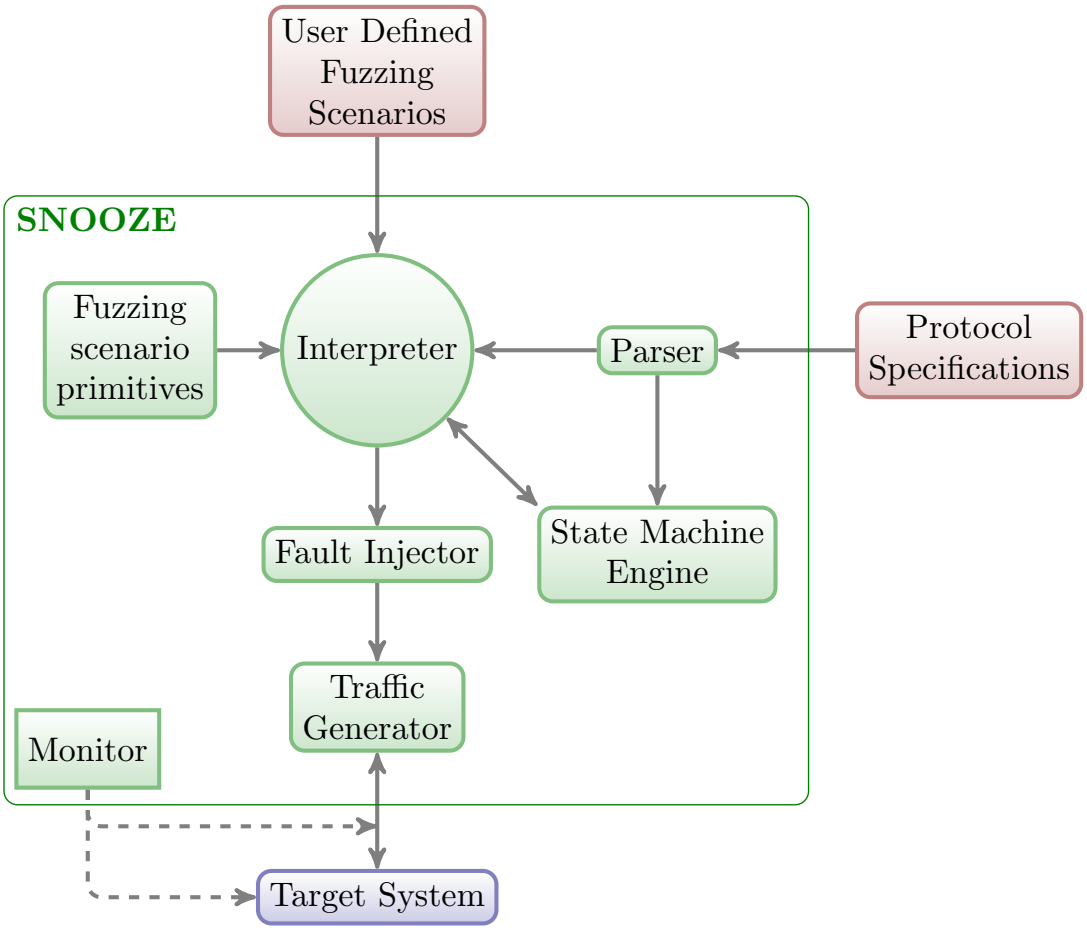


Figure 2.3: Detailed diagram of SNOOZE, illustrating its main components. Adapted from [2].

second to track its propagation and which variables are influenced by it during runtime. This enables the user to discover the flows between data sources and sinks. The main purpose of this information is to identify the most relevant paths of execution, which can later be used to improve the efficiency of the fuzzer. To calculate this metric, coverage analysis techniques are used. Another important component is a monitor to detect faults caused by the fuzzer. In addition this method should be able to automatically determine if faults are exploitable.

The authors do no present any results of using their methods in real applications. Nonetheless this paper proposes advancements to the field of taint analysis, in particular, for large-sized applications with a wide input domain. Therefore future work includes first, applying the approach to real targets and investigate its efficiency and second, integrate it in a comprehensive fuzzing environment.

One of the typical issues with fuzz testing is handling of checksums. In order to have meaningful coverage in protocols or file formats that employ checksums to verify integrity of inputs, it is crucial that the fuzzer can properly handle these. TaintScope [37] uses dynamic taint analysis and symbolic execution techniques to deal with those situations. It can inspect both Windows and Linux binary executables.

The paper identifies the three prime traits of TaintScope:

1. The program tries to identify checksum fields by branch profiling and alter the control flow to circumvent them.

2. In addition TaintScope uses dynamic taint analysis to uncover which bytes from valid inputs are used for system or library calls. Thereafter those bytes are fuzzed in the hope of triggering vulnerabilities. This way only the most promising data is fuzzed thus significantly reducing the mutation space.

3. Finally, the previous steps are performed by TaintScope in a truly automatic fashion. In particular, it leverages concrete and symbolic execution techniques to fix the checksums tests in the fuzzed inputs that was not able to bypass in the first step.

With this tool the authors discovered 27 previously undisclosed vulnerabilities in application such as Google Picasa, Microsoft Paint, Adobe Acrobat family, etc.

2.3.4. AUTOMATIC PROTOCOL INFERENCE

[38] investigates a different aspect of fuzzing: protocol specification. When developing fuzzers for network protocols expressing the protocol is often the most time consuming and error prone step. Typically it entails describing the different fields using a predefined set of primitives. In addition it requires access to the protocol definition which may not always be completely available. The main goal of this research is to present a system that automatically infers the state machine of the target (through reverse engineering) and generates inputs for a stateful fuzzer.

The paper lists four main contributions.

1. Identify various features to identify similar messages in a network session. This is necessary to select and cluster messages of the same type.

2. Present novel techniques to infer the protocol state machine.

3. Produce results from real applications where the state machines were successfully extracted.

4. Leverage the previous points to automatically generate protocol specifications and contribute such work to the Peach fuzzing platform.

The system developed by the authors was able to produce specifications for complex protocols such as SMTP, SMB, SIP, etc. In addition they successfully applied it to the Agobot malware and concluded that such a tool can be beneficial to malware analysts.

Peach [10], is often referred to as the most popular fuzzing framework and is capable of both generation and mutation based fuzzing. Its structure is quite versatile as it allows users to define modular components that can be used interchangeably in different fuzzers. To define the data format, type information and relationships in the data to be fuzzed, Peach uses manually crafted *Peach Pit* files. Due to the flexibility and maturity of Peach, it was chosen for the development of a smart meter fuzzer. Thus Section 3.3 is solely dedicated to this topic.

Despite the success and popularity of Peach there are a myriad of alternatives publicly available [39]. One of these is Sulley.

The Sulley Fuzzing Framework tries to learn from the limitations of other fuzzers and from Peach in particular, to make it easier to use and increase efficiency.

In [40] two researchers from Tipping Point, Pedram Amini and Aaron Portnoy present the reasoning behind this framework, and its main features and advantages over competing solutions. They claim that powerful frameworks suffer from a steep learning curve, while simpler tools quickly reach their limitations. In addition most fuzzers cannot be reused for different protocols and thus must be rewritten when the target changes.

Sulley introduces a simpler syntax with a block based data representation. In addition, Sulley fuzzers are written in Python which is considerably easier than C-like languages, which are typically used by other frameworks. To mitigate the difficulty in adapting fuzzers, Sulley supports the abstraction of complex types and the creation of commonly used helper functions, that can be programmed once and reused in different contexts.

Researchers at the University of Oulu in Finland have been working on black-box testing of the protocol implementations under the project PROTOS [41] and its follow-up the PROTOS Protocol Genome Project [42]. Particularly in the latter the focus is on automatically inferring a model that would subsequently be used to generate test cases.

The PERMU algorithm described in [43] was used as a proof of concept to asses the feasibility of this approach. The main principle behind PERMU is to build a domain model from an input data set through a nondeterministic finite automaton.

Still within the PROTOS Genome project, [44] concentrates on determining the best compromise in terms of efficiency and developer effort between fully random testing and human provided models. The idea is to identify structural building blocks common across most protocols that are typically poorly implemented and thus may lead to security flaws. They call these small structures protocol genomes, hence the project name. The authors were able to identify flaws using these techniques in three categories of programs: image parsers (GIF and JPEG), office packages (DOC, RTS and XLS) and security software (ZIP, RAR, etc.).

Despite the simplicity and limitations (*e.g.* no domain specific knowledge) of this approach it was effective as it was able to detect several issues with little effort from the security tester.

Viide *et al.* in [45] also focus on model inference. To accomplish it, they identify two necessary steps: first, choose a formal system for this model and second, implement structure inference. For the former they developed a domain specific language called "functional genes". And to express the structure the researchers chose to use Context Free Grammars (CFG) due to their ability to describes syntactic structure. To obtain the model they use training data that should provide enough coverage of the protocol under study. This approach aims to provide the best balance between truly random fuzzing (diminutive cost of entry but low coverage) and manual testing (labor intensive but provides best coverage).

After completion they apply the inferred model to a fuzzer. The software targets used to test the models were anti-virus programs. These are capable of parsing various formats, by definition process input from dubious sources, are usually run with administrator privileges and can be found on most computers, including those in critical networks. Five anti-virus programs were used with ten archive formats. For each format between ten to one hundred training files were used that were subsequently employed to generate up to 320 000 fuzzed files. Somewhat surprisingly they were able to find vulnerabilities in all but one of the software targets.

As conclusion, the previous results show that automatically inferred models can be effectively applied to reduce the cost of fuzz testing. However in the current form the provided coverage is still minimal. Nonetheless this avenue of research presents a lot of potential as even with all the aforementioned limitations it was able to identify flaws in targets that should

be the most resilient.

2.3.5. WHITE-BOX FUZZING

An alternative approach to fuzzing is entitled white-box fuzzing. [46] presents a comparison between black-box and white-box fuzzing. This technique aims to address one of the main limitations of the traditional methods, low coverage. As previously discussed, it is possible to improve coverage up to a certain degree however white-box fuzzing will often, due to its nature, outperform black-box in this regard.

The authors of [47] propose a methodology based on this alternative concept. First start with a well-formed input, then symbolically execute the program while recording any constraints on inputs from conditional statements that may appear. These constraints are then negated and solved with a constraint solver, yielding new inputs that will lead the program to different execution paths. This step is repeated for increased coverage using a novel heuristic algorithm. Everything comes together in a tool, SAGE (Scalable, Automated, Guided Execution) developed by the authors.

In a subsequent paper [48] the researchers present grammar-based white-box fuzzing to enhance white-box fuzzing. The constraints are generated from the symbolic execution of the target, via a dynamic test generation algorithm. This algorithm has two main components. The symbolic constraints are expressed in term of tokens returned by the lexer instead of symbolic bytes. And an adapted constraint solver suitable for symbolic grammar tokens. The solver searches for solutions that satisfy the constraints and the given grammar. They applied this concept to the JavaScript interpreter of Internet Explorer 7 and were able to obtain higher coverage, than with black-box fuzzing, while using less tests.

SAGE [47, 49] performs dynamic symbolic execution at the x86 binary level. In practice what this means is that one does not need access to the source code, which greatly increases the usefulness of this tool. On the other hand access to SAGE is limited to internal Microsoft projects, although there are some indications [50] that it may be released publicly in the future.

The choice between white-box or black-box fuzzing depends on various factors. Black-box is often simpler, lighter, easier and faster but has limited code coverage. Alternatively, white-box can yield better results but it usually involves more complexity, requiring additional time to develop and run. Ideally both should be used, the former to quickly discover the low hanging fruit and the latter to identify more sophisticated flaws, deeper within the

code.

The number of available fuzzers and fuzzing frameworks is quite significant as this area is of great interest to both white and black hats. The interested readers are referred to [51] and [52] for a comprehensive analysis of the most relevant frameworks available at the time of their publication.

2.3.6. IDENTIFYING EXPLOITABLE VULNERABILITIES

After the fuzzing process is complete it is important to analyze the results. The procedure may include going over the number of crashes or exceptions thrown by the target device to assess if they pose a security vulnerability.

Lu *et al.* [53] propose a technique to facilitate the automation of this important step, which is typically purely human. They introduce an "automatic fault localization" method for fuzzing as well as an "automatic vulnerability analysis system" titled FuzzLoc. The main objective is to save human effort, in other words, improve the efficiency of the process by identifying and categorizing key instructions that may cause exceptions.

The authors applied their tool to Adobe Reader and were able to discover previously know but also unknown (zero-day) vulnerabilities. FuzzLoc improved the efficiency of the fault localization process since the total number of instructions analyzed by humans was reduced. Notwithstanding some human guidance it is still crucial.

[54] proposes a procedure dubbed "fuzzing by weighting attack with markers" specifically developed to detect one of the most common class of vulnerabilities: buffer overflows. Autodafé is the name given by the author to the proof of concept tool used to assess the efficiency of this method.

Prior to explaining the routine it is important to define two key concepts. *Tracer* is a debugger that monitors and reports all the dynamically functions invoked by a program. This is useful to identify unsafe functions such as `strcpy`. The second concept is a *marker*. Every variable (*e.g.* a string) directly controlled by users is considered a marker.

The main idea behind weighting attack with markers is to identify unsafe functions that take markers as arguments. Autodafé does this by assigning a weight to each user controlled value (*i.e.* a marker). The value of this weight increases every time an unsafe function includes the marker in its arguments. Thereafter markers are tested in order according to their weights. With this technique it is easy to identify high potential markers hereby decreasing the variable space that needs to be fuzzed.

The results section of this paper is surprisingly sparse making no mention of the type or quantity of targets under study. It does however state that it discovered eighty known and 865 unreleased buffer overflow vulnerabilities in “modern” software.

3

BACKGROUND

3.1. COMMUNICATION PROTOCOL

The Device Language Message Specification (DLMS) consists of a general concept for abstract modeling of communication entities. On top of this, the *COmpanion Specification for Energy Metering* (COSEM) provides a set of standards that define the rules for data exchange between smart grid devices, such as an energy meter and a data accumulator. Together they enable several features, such as (a) an object model to view and access the different functionalities of a meter, (b) an identification system for all data, (c) a method for communicating with the model and (d) a transport layer to accommodate the information flows between the meter and other devices. This combination is known as DLMS/COSEM.

The protocol is based on the *Open System Interconnection* (OSI) seven-layer model. However they are practically collapsed in four - physical, data link, transport and application layers.

In a similar fashion to the OSI model, the physical layer in DLMS/COSEM defines how to transfer information to and from the meter. The data layer provides the messaging methods to modify data and communicate with the device. The transport layer enables data transfer based various interfaces. Finally the application layer represents the functional aspects of the energy meter so applications can access them.

The DLMS/COSEM metering specification is described in the DLMS User Association colored books:

- Blue book [55]: covers the COSEM interface classes and object model, as well as the standardized OBIS codes.
- Green book [56]: details the different protocols used to access and manipulate the COSEM objects.
- Yellow book [57]: describes the conformance requirement and procedures.
- White book [58]: contains the complete DLMS/COSEM glossary.

The summary of the protocol that follows is based on the aforementioned books, in particular the green and blue books. The reader interested in the complete details is advised to consult the original sources. The goal of this Chapter is not to reproduce the content of said references, but to provide enough background knowledge so the reader can fully understand Chapter 4 on eFuzz.

DLMS/COSEM can be divided in three major components:

1. Modeling: abstract representation of the meter functionalities in the form of objects.
2. Messaging: mapping of the services to application layer protocol data units (APDUs) and respective encoding.
3. Transporting: Physical-level communication channels used to carry messages between devices.

With respect to the specification, modeling is described in the blue book, while messaging and transporting are defined in the green book. It is important to note that in all communications profiles the COSEM application layer remains unchanged. However depending on the communication interface between client and server, APDUs are assembled differently.

3.1.1. COSEM

The smart meter has evolved from a simple recording device to a hub that depends on communication capabilities, interoperability and system integration. COSEM, aims to fulfill these requirements by proving a standardized method to convey energy measurements over a spectrum of connecting interfaces. It also provides the necessary functionalities that enable continuous remote access.

Object modeling techniques are used to facilitate access and manipulation of different types of data. Therefore one of most important parts of COSEM is to formally specify the

interface classes and objects used to accomplish such task. In particular to identify standard data items unambiguously in a vendor-independent manner, the OBject Identification System (OBIS) was created. The identification of each object is regulated and provides the unique mapping between the OBIS codes and the respective data items.

The standardized objects and interface classes can be modified by vendors to comply to national requirements or fulfill contract requirements over an entire range of products targeting different customers, for example domestic, industrial or commercial. This way it affords the necessary flexibility to adjust the product offerings according to the client's requirements without sacrificing interoperability.

COSEM INTERFACE CLASSES AND INTERFACE OBJECTS

An interface class (IC) is composed of attributes and methods and can be instantiated. These instances are denominated interface objects, or simply objects. Attributes define the unique characteristics of an object, *e.g.* the identification. And methods provide a way to access or modify the values of the attributes.

Similarly to object oriented programming languages, one of the main advantages of ICs is the abstraction they provide over objects that share certain characteristics. This aggregation eases the process of accessing and modifying objects.

To accommodate for the aforementioned flexibility vendors may implement proprietary methods and attributes to any object. Figure 3.1 depicts an example of an IC and its instances.

Metering equipment hosts a set of logical devices in a single physical device. Each of the devices enables a subset of the complete functionality supported by the hardware depending on the communication interfaces. These functionalities are accomplished through the use of COSEM interface objects.

The register is identified by the value of `logical_name` which contains an OBIS identifier. The `value` attribute holds the content of the register. Any COSEM meter is composed by a set of such objects that act as a gateway to all its capabilities.

APPLICATION ASSOCIATION

In DLMS/COSEM the communication model follows the server/client paradigm. The meter acts as a server, and replies to the client's application requests to retrieve data, change configurations, perform specific actions, etc.

Prior to gaining access to the COSEM objects in the server, both parties, client and server, need to define the context, which includes:

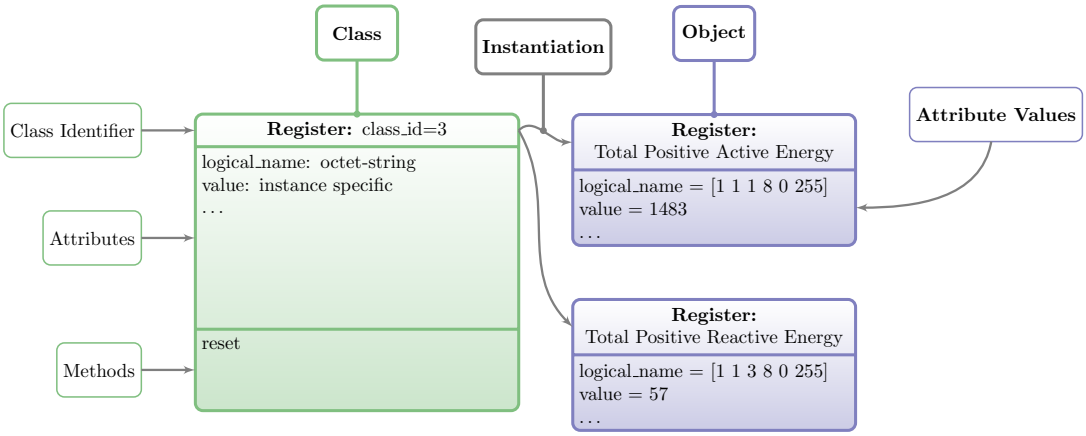


Figure 3.1: Example of an interface class with two instances.

- The application context.
- The authentication context.
- the xDLMS context.

xDLMS stands for extended DLMS and is a extension of the DLMS protocol, with emphasis on metering applications. In the context of this thesis, the differences between the two are not pertinent and therefore both terms are used interchangeably.

The exchange of this information is called an application association (AA). Depending on the AA, different access rights may be granted by the server. Permissions can be defined with respect to object visibility but also with access to specific attributes and methods. To that extent, the complete list of visible objects can be retrieved by the client and is called association view.

In order to enforce access rights, DLMS/COSEM defines security policies for the access and transport of data. Access controls restrict access to the data stored in the meter. While data transport pertains to the use of cryptography to protect the data in transit. Ideally, only the parties with the necessary keys can then decrypt the data and obtain access to the original content, the plain text.

DATA ACCESS SECURITY

Access to data can be restricted in DLMS/COSEM. Therefore metering equipment must authenticate the clients to ensure they are only awarded access to the data they have permission. The authentication context is negotiated between client and server at the AA stage.

DLMS/COSEM supports three categories of data access protection:

- Lowest level security (no security).
- Low Level Security (LLS).
- High Level Security (HLS).

Lowest level security is meant to retrieve basic information and it does not require any type of authentication.

Low level security requires the use of a password. The server is not authenticated by the client. This should only be used when the communication channel is protected and ensures sufficient protection against eavesdropping and or other type of channel alteration (*e.g.* replay or man in the middle attacks). The client transmits the password during the AA establishment.

High level security provides mutual (client and server) authentication. Similarly to LLS, the authentication takes place during the AA. The authentication process involves four steps that consist of exchanging challenges and inspecting the results with cryptographic methods. The HLS security context is indicated for all situations where no protection of the data communication channels is expected. This mode can use four different algorithms, MD5, SHA-1, GMAC (Galois Message Authentication Code) or a secret method known only by the meter and the client.

DATA TRANSPORT SECURITY

In addition to client authentication, the data transport can be encrypted. These protections are applied to APDUs to ensure they can not be deciphered on transit. The security policies available for data transport are:

- No security.
- Authentication.
- Encryption.
- Authentication and encryption.

Figure 3.2 shows how the various structures are used to encrypt and authenticate an APDUs.

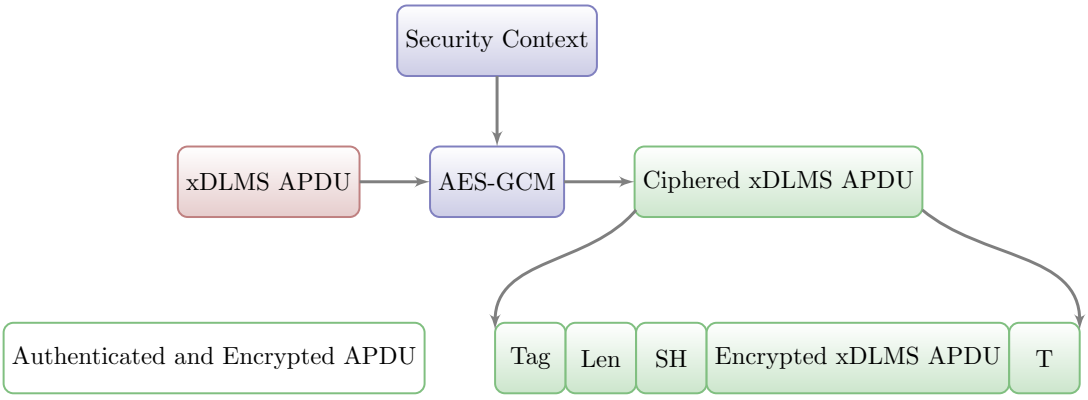


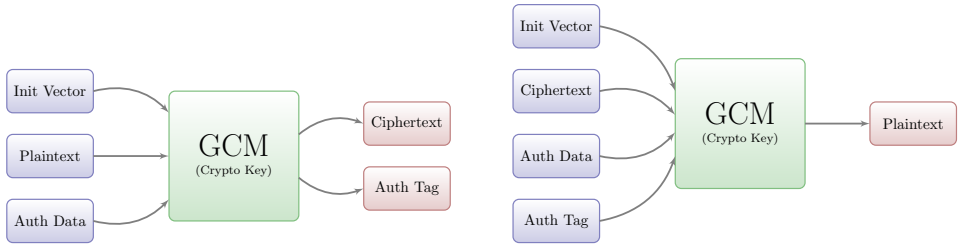
Figure 3.2: Example of an encrypted and authenticated APDU.

As Figure 3.2 shows, an encrypted and authenticated DLMS APDU is composed of five segments. First, the tag identifies the APDU type. Second, the length field is self explanatory. Third, the security header specifies which security policy for data transport shall be used. The fourth field contains the ciphertext. And finally, the fifth field, carries the tag used for authentication purposes.

For the policy with encryption only, the only differences is on the fifth field T, which is dropped. On the other hand, for the policy with authentication only, T stays but the ciphertext is replace by the plain text.

DLMS/COSEM only allows one security suite that is used for symmetric key cryptographic block ciphers: Galois Counter Mode (GCM) with AES-128. It uses a shared block cipher keys (EK), a shared authentication key (AK) and an initialization vector (IV). GCM was designed to provide both data authenticity and confidentiality. Section 4.2.1 explains in more detail the AES-GCM block cipher.

Figure 3.3 depicts what fields are required for encryption and decryption.



(a) Encryption (b) Decryption

Figure 3.3: Encryption and authentication of xDLMS Application Layer Protocol Data Units (APDUs).

The initialization vector is the concatenation of the system title and frame counter. The former is device specific and is announced in the configuration phases of the session. While the latter is transmitted alongside the ciphertext. The key for the Galois Counter Mode is specific for each device, in other words, both the meter and the PC have their own. Lastly, the authentication data includes the symmetric authentication key. The decryption process is similar but adds a verification steps for the authentication tag.

3.1.2. COSEM APPLICATION LAYER

The major part of the COSEM application layer is called the COSEM Application Service Object (ASO). It is used to provide services to the COSEM Application Processes and leverages the services provided by the layers underneath it. It is composed of at least three mandatory components on the client and server sides:

1. The Association Control Service Element (ACSE), whose goals are to establish, maintain and release application associations.

2. The extended DLMS Application Service Element (xDLMS_ASE), which provides data transfer services between different COSEM application processes. As explained above, xDLMS this is a backward compatible extension of the original DLMS protocol which focuses on metering applications through the use of COSEM interface objects.

3. The Control Function (CF). This function specifies how the ASO services invoke ACSE service primitives, xDMLS_ASE and other services of the supporting layer

Services are described in a standardized format to facilitate comprehension. Additionally they include the primitives and parameters that characterize each service. Each service may have one or more primitives that relate to the activity it performs. The service parameters convey the type of data associated with a service. Each service can incorporate zero or more parameters.

Services' statuses are identified with the assistance of four generic terms:

- REQUEST - primitive from user to the layer to request the initialization of the service.

- INDICATION - Primitive from layer to user to indicate an event of significance to the user. Typically it is used as a response to a previous request, but can exceptionally indicate internal events of the layer.

- RESPONSE - primitive passed from user to layer with respect to an indication primitive previously invoked.

- CONFIRM - confirmation primitive with direction layer to user to transmit the results of one or more previous primitive service requests.

3.1.3. THE DLMS/COSEM COMMUNICATIONS FRAMEWORK

As mentioned, DLMS/COSEM uses the client/server model for communications. The metering equipment assumes the role of server since its main function is to reply to requests from other devices (clients). Nonetheless unsolicited services are also available to the meter to allow notification of unexpected events.

Data exchange between server and client is accomplished by means of messages, APDUs. These data units are modeled as SERVICE.requests and SERVICE.responses. Application processes (APs) model the service requests and respective replies. Since most often server APs and client APs do not live in the same device these messages need to be adapted according to the interface connecting them and the respective protocol.

A typical communication flow would initiate with the client AP requesting a certain service and the server AP provides it as show in figure 3.4. The communication profile defines the supporting layers but the service primitives abstract over them. It is important to note that a client AP may exchange data with multiple server APs concurrently, and vice-versa.

INTEROPERABILITY AND INTERCONNECTIVITY

Before proceeding to the description of the different communication profiles available in DLMS/COSEM it is important to define interoperability and interconnectivity in the context of APs. Interoperability and interconnectivity ensure devices from different manufacturers can communicate with one another via DLMS/COSEM.

Interoperability relates to the application layer. A client AP is interoperable with a server AP if they are able to establish AAs. On the other hand, interconnectivity is a protocol level concept. For both parties' APs to be able to successfully communicate they must be interconnectable and interconnected. Therefore interconnectivity is the ability of the COSEM AP to establish a connection across all layers.

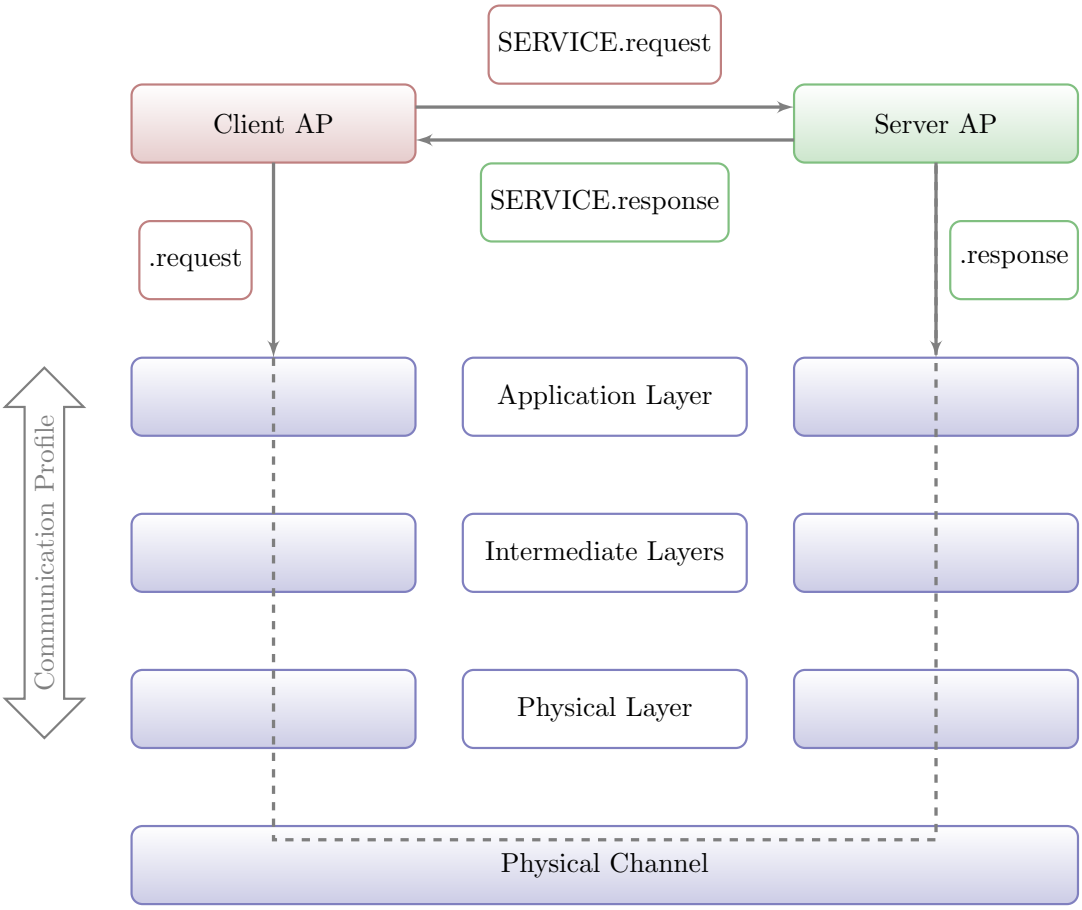


Figure 3.4: Client/server relationship and protocols with various communication profiles.

3.1.4. COMMUNICATION PROFILES

Communication profiles transcend single protocol layers and describe how to communication with COSEM APs. In fact, the COSEM APs rely solely on the application layer. They are independent of the communication profile and are the only ones containing COSEM specific elements. These are called Extended DLMS Application Service Elements (xDLMS_ASE) and provide access to the COSEM interface object services. These services are independent of the lower layers.

The communication profiles are defined by the number of protocol layers included, their parameters and if they are connection oriented or connectionless. A single metering equipments often supports multiple communication profiles, for example IP or PLC. Since clients are often more restricted with respect to the interfaces available, they choose which one is used.

Figure 3.5 shows the communication profiles defined in the green book.

The first supports data exchange via an optical or electrical port, PSTN or the GSM network. It consists of 3-layers, connection oriented and based on HDLC.

The second is based on the Internet standards TCP-UDP/IP. Therefore traffic can flow over any physical media that supports Internet communication such as Ethernet, GPRS, etc. This is a connectionless profile and includes a wrapper in addition to the transport layer TCP or UDP.

Finally the S-FSK PLC profile supports S-FSK modulated communication over power lines. The transport layer can be the connectionless LLS or the LLC sub layer in addition to the HDLC protocol.

The fuzzer described in Chapter 4 uses the communication profile shown in green tones. Therefore the details of such profile are described in more detail than the remainder.

The xDLMS protocol is communication oriented which means a session needs to be established between the client and the server before any requests can be placed.

There are three phases in this process:

1. An application association is established between client and server APs. The lower layers (physical and data link) must already be connected so that an application level PDU can be transmitted.

2. Data exchange can take place provided by the xDLMS service elements.

3. When data exchange is concluded a request for disconnection is sent by one of the parties (most often the client). The AA is then released.

POWER LINE COMMUNICATION

The DLMS/COSEM S-FSK PLC communication profile is designed for use over the electricity grid network. This communication has limited range and typically is performed between household meters and data concentrators on substations to aggregate the data from a subset of metering equipments. It is thereafter communicated upstream via other communication channels.

In the Power Line Communication (PLC) medium the client role is fulfilled by the data accumulator while the server is performed by the meter.

This communication profile uses two sub layers for the data link layer: the Medium Access Control (MAC) and the Logical Link Control (LLC). The MAC sub layer assumes respon-

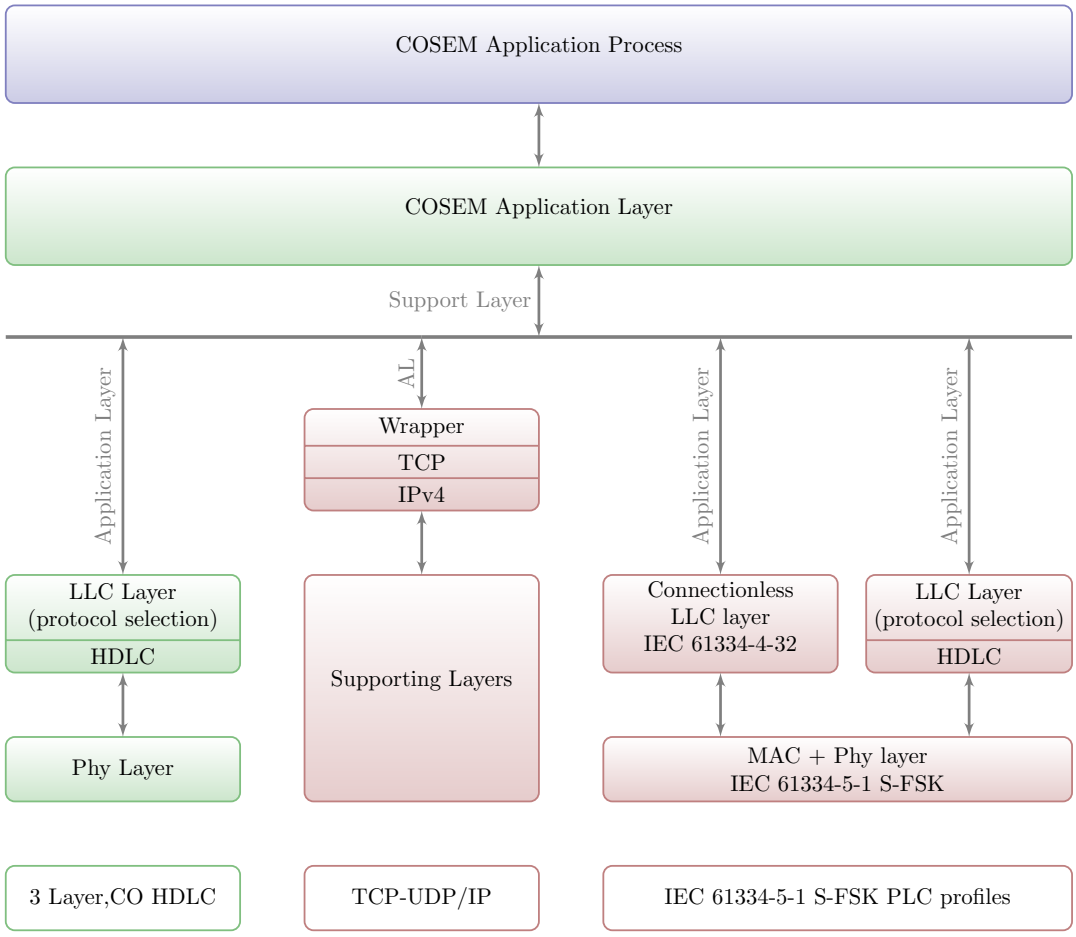


Figure 3.5: DLMS/COSEM communication profiles

sibility for controlling access to the physical lines and physical device address. The LLC sub layer controls the logical links.

The LLC sub layer can assume one of two forms:

- Connectionless LLC sub layer
- LLC sub layer using the HDLC based data link layer

IP CONNECTIVITY

DLMS/COSEM also supports IP based networks. In particular it supports two modes. For connectionless operation it uses the User Datagram Protocol (UDP), while for connection-oriented communications it uses the Transport Control Protocol (TCP). These protocols are covered in the respective standards however DLMS/COSEM defines an additional sub layer,

sitting between the TCP/UDP layer and the the COSEM application layer dubbed wrapper to enable their use. Due to the use of IP layers a DLMS/COSEM application can live along other Internet based applications such as HTTP or FTP clients.

The wrapper fulfills two main functions:

- Additional addressing capabilities.
- Length of data transported. This feature is useful when an APDU is split among various TCP packets. It is then trivial to identify the reception of a complete APDU.

DIRECT LOCAL CONNECTION

Direct local connection typically accommodates devices such as hand held units used by technicians for on-site visits or equipment installations. They use optical serial communications to interface with the meter.

Direct communication requires an initial serial handshake, the procedure of which is shown in figure 3.6.

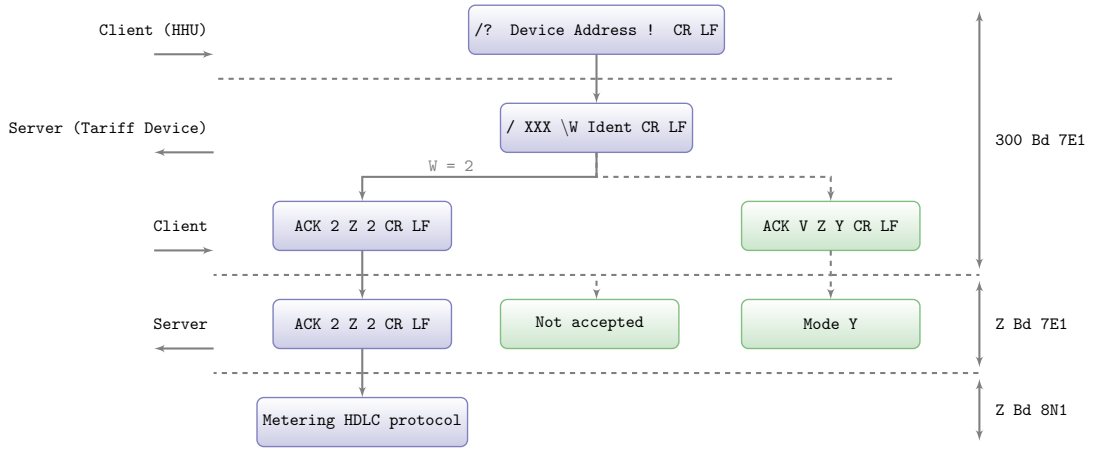


Figure 3.6: Entering protocol mode E (HDLC)

First the client sends a request with the following message `/? Device Address ! CR LF`. Obviously the device address changes between meters. This transmission is performed with a baud rate of 300 bps, 7 data bits byte, even parity and one stop bit (7E1).

The server acknowledges the received message with the following `/ XXX \W Ident CR LF`. Where `XXX` is a 3-byte ASCII representation of the manufacturer identifier, and `Ident` identifies the equipment model. If `W` equals 2, the protocol mode E is chosen, which is colored blue in Figure 3.6. However, older devices that do not support this mode, switch over to

protocol mode C until terminated, shown in green. Only the former is of interest to fuzzing, thus for the remainder of the document it is assumed that only protocol mode E exists.

After, the client sends with the following message `ACK 2 Z 2 CR LF`. Where Z is the same Z indicated by the server. The mode identified by Z is then enforced for all forthcoming data exchanges. Z is used to determine the serial configuration the meter would like to change to for the remainder of the connection. For example a 5, means 9600 baud rate, 8 data bits, no parity and 1 stop bit. The server repeats the message as acknowledgment.

From this point onward the HDLC metering protocol can now take over.

3.1.5. DATA LINK LAYER

Due to the fact that DLMS/COSEM supports both connection-oriented and connectionless communications, the data link layer is divided in two sub layers: Medium Access Control (MAC) and Logical Link Control (LLC). These sub layers are specified with respect to service specifications and protocol specifications.

PROTOCOL SPECIFICATION FOR THE LLC SUB LAYER

The LLC protocol data unit is shown in figure 3.7.

Figure 3.7: LLC format as used in DLMS/COSEM.

The value for the `Destination_LSAP` (Local Service Access Point) is fixed at 0xE6. While the the value of the `Source_LSAP` is 0xE6 (for command) or 0xE7 (for response). The `LLC_Quality` field is reserved for future by the DLMS User Association and must be set always to 0x00. Finally the information field carries the PDU.

MAC SUB LAYER

The MAC sub layer uses the HDLC frame format depicted in Figure 3.8.

Figure 3.8: MAC sub layer frame format.

The flag field is always one byte long and assumes the value 0x7E. If multiple frames are sent continuously a single flag is used to terminate and initiate the subsequent frame.

The frame format field has a length of two bytes and is composed of three subfields, format type, segmentation bit and frame length sub-field. The format type (4 bits) is set to 1010 (binary). The segmentation field indicates if an APDU is split in various frames. In such a situation this bit is set to True (1) for all frames that contain the single APDU except the last one. Finally the last subfield is the frame length which encodes the length of the frame excluding opening and closing flags in 11 bits.

The control field occupies one byte and indicates the type of command or response as well as sequence numbers carried by the frame. Table 3.1 enumerates the possible values for this field. The following section goes into more detail on the different types of frames available.

The Header Check Sequence (HCS) computes a checksum based only on the header of the frame, excluding the opening flag. It spans over two bytes. If frames do not have an information field, the HCS is not included in the frame.

The information field can assume any size and carries the payload.

Finally the Frame Check Sequence is very similar to the aforementioned HCS but computed over the complete frame, excluding the opening flag. It is also two bytes long.

Figure 3.9 features an high-level aggregate view of the different layers that make up a DLMS/COSEM request over serial port.

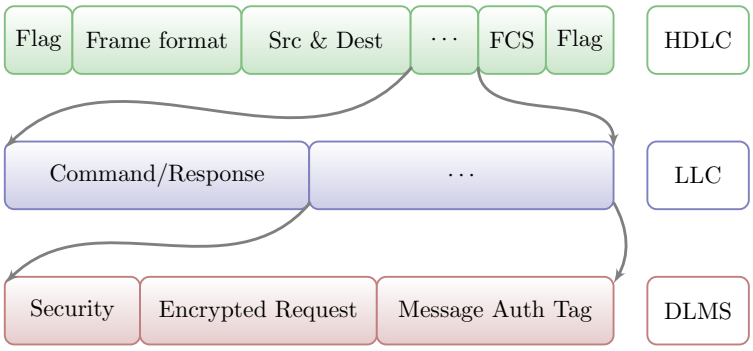


Figure 3.9: DLMS/COSEM packet for direct connections.

As stated when using a direct connection in DLMS/COSEM, the High-Level Data Link Control (HDLC) protocol is used. This data link layer, show in green, is composed of opening and closing flags, the type of frame, source and destination identifiers, payload and checksum (Frame Check Sequence or FCS) to ensure data integrity. The Logical link control (LLC)

Command	Response	Encoding
I	I	R R R P/F S S S 0
RR	RR	R R R P/F 0 0 0 1
RNR	RNR	R R R P/F 0 1 0 1
SNRM		1 0 0 P 0 0 1 1
DISC		0 1 0 P 0 0 1 1
	UA	0 1 1 F 0 0 1 1
	DM	0 0 0 F 1 1 1 1
	FRMR	1 0 0 F 0 1 1 1
UI	UI	0 0 0 P/F 0 0 1 1

Table 3.1: Control field bit assignments of command and response frames

explained above, colored blue, is the upper sub-layer of the data link layer. It is short in length and is used to specify is the payload is a response or a command. The application layer, tinted red, contains the xDLMS protocol data unit. Its content specifies the level of security it is using and, if applicable, proceeds to transmit the encrypted request followed by the authentication tag.

FRAMES

The encoding of the command and response control fields, shown in Table 3.1, is used for the control field of the MAC sublayer.

RRR is the receive sequence number N(R), SSS is the send sequence number N(S) and P/F is the poll/final bit.

The I frame, or information frame is the most generic and shall be used to transfer information between client and server.

Received ready or RR, indicates the server is ready to receive an information frame or to acknowledge previously received I frames. It can be used after RNR (received not ready) to signal the previous busy conditions not longer apply.

The set normal response mode (SNRM) command is sent by the client to indicate to the server all control fields shall be one octet in length, which corresponds to the normal response mode (NRM). To accept the change, the server responds with a UA (unnumbered acknowledge) frame.

The disconnect command (DISC) is used to request the termination of a previously es-

tablished session. It is initiated by the client to signal it is about to suspend operations. The server acknowledges the request with a UA frame.

The disconnected mode (DM) response is used by the server to signal it has lost the logical connection to the data link. No further commands are accepted until the receipt of a mode setting command (SNRM).

Frame reject (FRMR) response is used by the server to signal one of the following anomalous conditions has occurred:

- Command or response that is not define or not implemented.
- I/UI command or response with an information field exceeding the maximum allowed length.
- Wrong sequence number for an I frame.
- Frame containing an information field when no such field is permitted by the control field.

Furthermore, these errors are not correctable by retransmission of the frame by the client.

Finally the Unnumbered information (UI) sent as command or as response does not affect the receive sequence number or the send sequence number. It is used to send information by the client or the server.

DESCRIPTION OF THE PROCEDURES

After establishing a physical connection but before establishing an active data like devices are in is the Normal Disconnected Mode, NDM. In this mode no information or supervisory numbered frames are transmitted or accepted. The server side is capable of accept and response to SNRM commands, accept and transmit UI commands, and respond with a DM response to DISC commands.

When the MAC connection is established the Normal Response Mode is activated. The server after receiving permission from the client (Poll bit set) can initiate response transmissions. The final response transmission in indicated by the server (with final bit set). Then it stops transmitting until receiving permission from the client.

The client is responsible for the initialization of the HDLC link. It does so by sending an SNRM command. The server replies with an UA response. If received correctly, the link is successfully established. Otherwise, the client can retry to establish a link at a later stage.

In addition the SNRM/UA message exchange is also use to negotiate data link parameters:

- the Maximum information field length (default is 128 bytes, the maximum varies depending on the physical medium).
- the Window size (the default is 1, and the maximum is 7).

Thereafter to disconnect the link the client sends a DISC command. The addressed server responds with a UA response at its first opportunity and enters the NDM mode. If at the moment of DISC reception the server is already in NDM mode it sends a DM response. The client after receiving a UA or DM message as response to a DISC command shall proceed to logically disconnect the data link.

3.1.6. COMPLETE FLOW

To assimilate the information presented in this section it is pedagogical to provide an example. It succinctly describes the complete communication steps that a simple request over a direct connect entails. Figure 3.10 shows the respective state diagram.

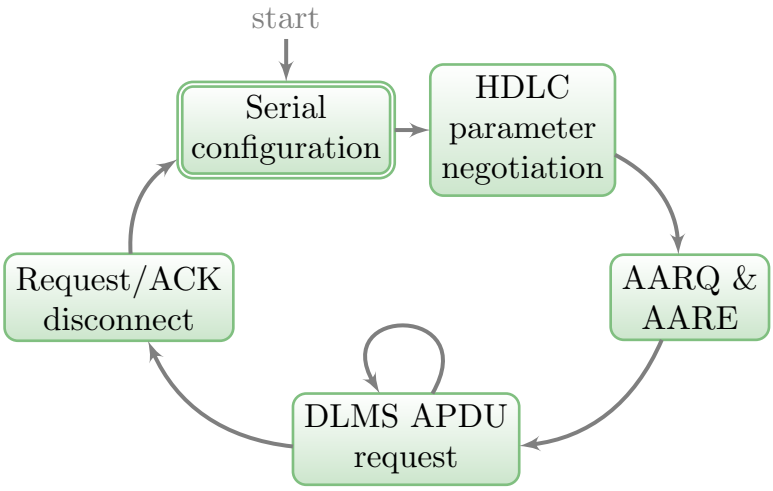


Figure 3.10: A complete DLMS/COSEM flow including establishing a connection, making a request, reading the response and closing the connection. This example is for the physical interface.

Serial connection is divided in three distinct phases. First, a handshake takes place where the meter identifies itself and announces the serial configuration it supports, including baud rate and other serial related parameters.

Second, the client (PC) initializes the HDLC (High-Level Data Link Control, a data link layer protocol) link by transmitting a frame to the meter. After reception the meter replies

with the frame configurations it supports. The third and last step to complete the connection establishment is the pair AARQ/AARE (Application Association Request and Response, respectively). The former is sent by the PC and the latter is the response of the meter. This exchange is used to establish more parameters specific to the DLMS layer (*i.e.* independent of the outer-layers which depend on the interface). If defined in the security configurations, these are the first packets to be encrypted and authenticated.

Finally, we are able to start the actual DLMS/COSEM communication. In this session, the client may query the server for usage data, time and date, or modify the parameters of the meter, such as increase the number of available tariffs, clear the logs, etc.

The last step is requesting the end of the session. This is done by sending a DISC frame.

3.2. FUZZING

Fuzzing is one of the premier techniques to uncover vulnerabilities through testing [5, 36] as it does not require access to the source code of the target and it has a low cost. The most common targets include network protocols and file formats.

Fuzz testing is a security technique used to detect bugs in software applications or network devices. It is an automated process that feeds the target with malformed data to induce malfunctions. In other words, it consists of trying numerous combinations of inputs, violating protocol or file format rules, to see how the target responds to unexpected requests. To detect these erroneous actions, fuzzing includes monitoring activities like memory consumption inspections, thrown exceptions, crashes or other unexpected behaviors.

Peach [10], the brainchild of Michael Eddington, has become one of the most mature and, arguably, the most widely used fuzzing framework. Hence this framework was chosen for the development of eFuzz. Peach is capable of both generation and mutation based fuzzing. To define the structure, type information and relationships in the data to be fuzzed, Peach uses manually crafted Peach Pit files. These files describe the protocol model and define how such model will be used to test the target.

The modular architecture of Peach encourages code reuse and provides easy extendability. The main advantages of using Peach, over of developing a fuzzer from the ground up, are the abstraction of the mutation strategies, the simple way the protocol can be modeled and the modularity of the various components (custom or built-in).

Fuzzing frameworks typically target software applications running on traditional computing platforms, such as desktops. To monitor the working state of the targeted software,

they might for example attach debuggers to the process or inspect memory consumption. Unfortunately, in embedded systems like smart meters, these capabilities are not always available and the fuzzing framework is usually physically separated from the investigated target. Therefore, to fuzz an embedded system like a smart meter, it is necessary to use a framework that supports the communication protocols available in the target and that allows specific types of monitoring developed for limited devices.

In general, fuzzers can be divided in three categories with respect to how they generate new packets or files. The first one is to feed completely random data as input to the target. This method is simple but is less likely to yield good results. In the second category, the mutation-based fuzzers alter pre-generated valid inputs in hope of discovering a vulnerability. In the third category, generation-based fuzzers create files or packets from scratch with the same intent based on a user specified model.

Compared to others, the main advantage of the second category is that little knowledge of the file or protocol format is required and thus the preparation time is modest. However the code coverage is dependent on the coverage provided by the original valid inputs. On the other hand, constructing new files or packets can only be done if the tester has sufficient knowledge of the file or protocol format. But this approach results in a more comprehensive test case suite and thus, it is more likely to find vulnerabilities. eFuzz, the fuzzer developed during the course of thesis and describe in Chapter 4, belongs in this category.

Table 3.2 provides a succinct summary of these advantages and disadvantages.

3.2.1. BASIC CONCEPTS

[59], a comprehensive book on fuzzing, uses the following definition: “a method for discovering faults in software by providing unexpected input and monitoring for exceptions”. One of fuzzing main advantages is that it can be run autonomously or semi-autonomously thus requiring little human supervision.

Fuzzing involves successive attempts of data manipulation and modification, followed by supplying this data to a target for processing. The most common scenarios where fuzzing tools are used are file formats or network protocols. In the first case the input that is fuzzed are the files themselves prior to being opened by a software application. While for the second the protocol packets are interfered with before reaching the target application or network device.

The other part of fuzzing, as stated in the previous definition, is the monitoring compo-

Type	Advantages	Disadvantages
Random	Simple. Quick. Low cost.	Attacks only the surface. Useless with checksums. Poor coverage.
Mutation	Relatively Simple. Reusable for different SW.	Needs numerous valid inputs to get a good coverage.
Generation	Efficient (if well modeled).	Time-consuming. Requires knowledge of the file/protocol format. Reusable only with the same format.

Table 3.2: Advantages and disadvantages of the different fuzzing approaches. Adapted from [5].

nent. In order to detect whether the fuzzed input has a nefarious effect on the target it is necessary to observe its health. There are various methods to accomplish this depending on the type of target. If the target is a software application it can be inspecting its memory consumption, attach a debugger to catch exceptions, monitor CPU usage, etc. On the other hand, if it is an embedded system it can be based on JTAG information.

In setups where no monitor can be run on the target machine, monitoring activities can consist of analyzing the communications, such as making sure the responses are valid, timing the latency for unusual delays or inspecting power consumption. In essence, any metric that may indicate the target is behaving in an anomalous manner.

The fuzzing process can be divided in various phases:

1. Identify target: selecting a particular application or specific file or library within an application.

2. Identify inputs: enumerating input vectors is pivotal to the success of fuzzing since most exploitable vulnerabilities are caused by applications accepting and processing unsanitized user input, or applying validation routines.

3. Generate fuzzed data: use predetermined values, mutate existing data or generate data dynamically (automated process).

4. Execute fuzzed data: *e.g.* sending a data packet to the target, opening a file, launching a target process (automated process).

5. Monitor for anomalous behavior: use any metric that is able to detect performance is deviating from normal. This help the fuzzer pinpoint which attempt was the root cause.

6. Determine exploitability: after identifying a fault it may be necessary to determine if the uncovered bug can be further exploited. Typically this is a manual process that requires expert security knowledge.

Fuzzing also has some limitations, in particular with respect to the type of vulnerabilities it can not detect. Some of these are:

- Access control flaws (*e.g.* checking correct user permissions).
- Poor design logic (*e.g.* allowing access without authentication since it is just a design bug).
- Back doors.
- Memory corruption (if they do not lead to crashes, *e.g.* when they are handled by the application by spawning a new process).
- Multistage vulnerabilities (fuzzing identifies individual flaws but cannot identify a multi-vector attack that chains together minor vulnerabilities).

3.2.2. FUZZING METHODS

Within the fuzzing categories laid out in the first paragraphs of the Chapter various techniques can be used. The choice of the best technique is completely dependent on the type of target and investment and commitment available for the fuzzing effort.

In its most basic form fuzzing can simply be based on random inputs, which consists of sending pseudo-random data at the target. It is the least effective method but can be used as a quick once over program to determine if the target has extremely fragile code.

The next option is manual protocol mutation testing. As the name suggests in involves no automated fuzzer. After loading up the target application the researcher simply enters inappropriate data in an attempt to crash the target or induce some undesirable behavior. This practice is capable of leveraging past experience and gut feeling during the audit with very minimal setup. In contrast, the obvious limitations pertain to the amount of labor involved.

As a consequence, manual mutation testing is slow and expensive. Nonetheless it is capable of uncovering more sophisticated flaws in the target, than the previous, but it is highly dependent on the tester's expertise and luck.

Another labor intensive system is based on pre-generated test cases. The first step involves studying a particular specification to understand data structures and acceptable value ranges. Then the researcher can create a set of hard coded packets or files that test boundary conditions or violate the specification altogether. On the bright side the test cases can be used to uniformly test multiple implementations of the same protocol or file format across different targets. However fuzz testing is bound to be limited as there is no randomness in the set. Once again this form of fuzzing is time consuming as it requires understanding the specification and manually writing the test cases.

The first automated method is called mutation or brute force testing. The starting point is a valid sample (or set of samples) of a protocol or data format which is automatically mutated. The mutation can be performed with various degrees of granularity, for example the scale of bits, bytes, words, etc. Mutation is a great early approach as it requires very little up front research and it is straightforward to implement. Nevertheless it can be inefficient as many CPU cycles are wasted on data that cannot be interpreted in the first place and is immediately dropped. As an example, checksums are notoriously problematic for mutation based fuzzers. Furthermore code coverage is dependent on the collection of known good packets or files that are tested. Since most protocol specifications or file definitions are relatively complex it takes numerous samples to get a decent coverage. However these challenges and limitation are offset by the fact that the process can be fully automated.

Finally, the most sophisticated technique is automatic protocol generation testing, an evolved version of brute force testing. Up front research is required to understand and properly interpret the protocol specification or file definition. The researcher shall then create a grammar or model to describe the protocol. This model identifies portions of the packet that are supposed to remain static, or are computed based on other fields, and others that represent freely fuzzable variables. Thereafter the fuzzer dynamically parses these templates, generates fuzzed data and sends the resulting packet or file to the target. The success of this approach is dependent on a researcher's ability to pinpoint those portions of the specification that are most likely to lead to faults in the target application during parsing. The downside is the time it takes to generate the grammar or definition. The fuzzer described in Chapter 4 belongs in this class.

3.2.3. FUZZER TYPES

Each target lends itself to its own class of fuzzer. The types of fuzzers available can be grouped in two larger sets: local fuzzers and remote fuzzers. Intuitively, in local fuzzers both the target and the fuzzer run on the same machine, while remote fuzzers involve listening on a network interface.

LOCAL FUZZERS

Locals fuzzers can be further refined in relation to how they work. Command-line fuzzers target applications that receive input from command line arguments. In this situation fuzzing consists of experimenting with malformed arguments. As described in Section 2.3 the original fuzzers were of this type.

A similar variation is called environment variable fuzzers. For this type, the researcher modifies the user-defined environment variables, for example `$HOME` or `$PATH`. Despite the simplicity, this approach can succeed when the developers of the target application overlook environment variables as user controllable values.

Another instance of local fuzzers are file format fuzzers. As already explained, the input for the target includes a set of user-defined malformed files, which are consequently launched using the target application in an attempt to cause anomalous behaviors. This is one of the most common techniques to fuzz test proprietary application such as the Microsoft Office Suite or Adobe Reader.

REMOTE FUZZERS

Remote fuzzers make up the second major class of fuzzers, they target software that listens on network interfaces. Remote fuzzers can be further discriminated in network protocol fuzzers, web browser fuzzers and web application fuzzers.

Network protocol fuzzers are dedicated to network devices. Targets include web servers and routers.

Web browsers are extremely complex applications that support a myriad of file formats and protocols. Moreover they are ubiquitous and are often the most used application in a computing device. From the perspective of an adversary they are also one of the easiest gateways to victims due to the Internet connection. As a consequence they are an important target for fuzzing. This subclass of fuzzers is predictably referred to as web browser fuzzers.

A similar target that, from the point of view of attackers, shares many of the same appealing attributes are web applications, such as Gmail or Google Docs. They have also grown

in complexity and regularly process user data, making them a suitable target for fuzzing. Fuzzers for this targets are designated web application fuzzer.

OTHERS

In addition to remote and local fuzzers, there are novel approaches to fuzzing that belong in their own class or are flexible enough to span over these two. In memory fuzzers are capable of taking a snapshot of a running process and then inject faulty data into one the parsing routines. Another example is fuzzer frameworks or fuzzer libraries. These are not traditional fuzzers but instead applications that simplify the creation of fuzzers. Frameworks provide, at the least, a syntax for representation of different types of data and can be readily adapted to support a variety of targets. Section 3.3 describes a framework, Peach, which is leveraged to fuzz a DLMS/COSEM smart meter.

3.2.4. DATA REPRESENTATION

Communication protocols define a set of rules to facilitate communication across devices. To model a protocol in a fuzzer, or in a fuzzing framework, it is important to understand those rules and what components they include as well as the most common types of protocols available.

FIELDS

Protocol rules are defined in a specification document which describes how individual components, know as fields, are separated, ordered and chosen. Protocol delimitation assumes three forms: fixed length fields, variable length fields and delimited fields.

Fixed length fields, as the name suggests, predefine a set number of bytes to be used by each field. These type of fields are common in headers of network protocols, such as IP, TCP UDP, and DLMS. Fixed length fields are useful when a highly structured and repetitive format is beneficial, which is the case for headers.

Variable length fields address opposite needs, when the length of the content is not predictable. This is often the case for media files or web pages. These fields provide protocol developers the flexibility required to accommodate a broad spectrum of use cases. Typically, variable length fields are prefixed by headers that indicate what type and size to expect.

Finally, protocols or file formats that use delimited fields are capable of storing data separated with specific delimiter characters. An example is HTML which uses angle brackets ‘<’ and ‘>’ to enclose the different tags.

Due to the ubiquity of protocols, many share common elements:

- Name value pairs.
- Block identifiers: values that identify the type of data being represented in binary data. Might be followed by variable or fixed length data. Fuzzing can be used to identify undocumented block identifiers that might accept additional data types, which in turn can also be fuzzed.
- Block sizes: generally consist of data such as name-value pairs that are preceded by one or more bytes detailing the type of field and the size of the variable length data that follows.
- Checksums: some file formats embed checksums throughout the file to help applications identify data that might be corrupted or invalid. Although checksums are not a security measure, they can impact fuzzing results as applications will generally abort file processing in the event of incorrect checksums. Thereafter it is meaningful fuzzers take the checksums into account and recalculate them based on the appropriate data.

TYPES

The simplest protocols are encoded in plain text and are designed to be readable by humans. They generally are less efficient than binary due to higher memory consumption. For example, FTP is a plain text protocol and since it is human readable can be handled manually using command-line tools. The readability also makes it easier to debug than binary.

Binary protocols are more common, in particular when considering complex protocols. They are optimized for inter-machine communication, not for humans, and thus are more difficult to manually decode. Without a deep understanding of the protocol the packets will not be particularly meaningful.

Network protocols are used in a myriad of applications, in particular for Internet communications. Examples include data transfer, routing, email, streaming media, instant messaging, among others. Similarly to protocols, file formats also specify a well defined set of rules for communicating with machines. Both network protocols and file formats can be encoded in binary format or in plain text.

3.3. PEACH

Peach is an open-source fuzzing framework, capable of doing both generation and mutation fuzzing. It has a modular architecture which not only encourages code reuse, but also makes it easily extensible; hence one can easily write custom mutators, fixups and mutation strategies. Moreover, it has features like agents, monitors and data analyzers, which make Peach a comprehensive tool to carry out fuzzing. Additionally it allows for the configuration of distinct fuzzing runs with specific publishers, logging interface, unique state machines, etc.

Peach provides various benefits over writing a fuzzer from scratch. It makes development notably faster as it abstracts the actual mutation of individual protocol or file format fields. The user is only required to provide a model for the packets or files. Furthermore, for common protocols it may not even require programming skills, as many standard components are already included. Extensibility and reusability are other relevant features of Peach. They ease the process of improving and expanding already existing fuzzers and development of common modules can be shared across multiple targets. Finally, in Peach it is possible to use deterministic strategies that make test suites repeatable. The importance of repeatability can not be overstated since it crucial to narrow down the root cause of failures.

3.3.1. PEACH PITS

Pits are at the core of Peach. They are the standard way to define a protocol packet or file format. Pits are XML files and can contain the following top level elements or sections:

- General Configuration - references external modules, sets global attributes, imports custom code, and configures the path of python modules.
- Data Modeling - expresses the data block structure of the fuzzed protocol or file format.
- State Modeling - defines the state machine used by the protocol under test.
- Agent - outlines the processes that are employed for monitoring, inspecting memory consumption, attaching debuggers to the target, etc.
- Test configuration - specifies the configuration for state models and agents, loggers, among others.
- Run setup - selects which tests are executed.

All of these elements can be used individually in various contexts or combined as necessary in different Pit files. The rest of the Section will delve deeper into how to write each top level element. For detailed information, the official documentation [60] along with the source code are the definitive sources.

3.3.2. DATA MODEL

Data models are the sections in the Peach Pit that describe the structure of the data, be it a TCP packet, or a CSV file. This element may contain type information, data relationships, static fields, etc.

Data models can be reused and referred by others. Referrals are analogous to inheritance in object oriented programming since the sub-model assumes all the fields defined in its super-model. This capability to inherit from other data models allows the user to break up complex data definitions into more readable portions. When a data model refers another one it copies all its elements, but they can also be individually overwritten to adapt to the user needs.

Peach primarily uses four elements for representing data: strings, numbers, flags and blobs.

String The string element defines a single or double byte string and is used to indicate text, or other human readable data. In addition, it can be used to represent strings composed of numbers with the use of the `NumericalString` hint. In this situations the Peach engine will supplement the ordinary string mutators with all the numerical mutators. It is also possible to specify a default value, whether it should be fuzzed, if it is a token[1], among other options.

Number The Number elements defines a number of lengths 8, 16, 24, 32, or 64 bits. In addition to the same options available for strings, it is also possible to define the endianness (the byte order of a number) and the number representation (*e.g.* hexadecimal).

Flags Flags are a data container that aggregates individual flag elements, which represent single bit fields. Individual flag elements can assume various formats (*i.e.* hexadecimal, string, or literal), and span an arbitrary range of bits.

[1] In Peach, tokens are data elements that should always exist and must be identified before continuing with the rest of the data block. If a token is not present Peach moves on to the next available block.

Blobs Blob elements are applicable when the user is unaware of the proper type definition for a certain field. They should be used as a last resort as Peach can only perform generic mutations such flipping bits or sliding specific data values through the blob, one byte at a time.

Blocks In addition to the four essential elements just described, Peach supports additional, more sophisticated elements, without which would not be possible to fuzz any non-trivial protocol. Blocks work as a container for other elements. Similar to data models they can also refer and be referred to. Furthermore they improve the readability of pit files since complex data definitions can be separated in different elements or reused.

Choice Elements Choice elements are similar to switch statements in programming because they allow users to choose only one block from a set of possible blocks. The branch condition is implicit in the order of the blocks since Peach will choose the first that matches the supplied data.

Relations Relations are important elements in Peach as they express relationships between data types. There are four types of supported relations: Size-of Relation, Count-of Relation, Offset-of Relation and When Relation. Size-of and Count-of relations are relatively straightforward to understand. A Size-of relation will be replaced with the size of the specified data element in the format chosen by the user. This type of relationship is very common in the headers of protocols which use variable-length fields. A Count-of relation counts the instances of a certain element. The next relation, Offset-of relation, represents data with specific offsets between various elements. Finally, a When-relation evaluates a user provided python boolean expression to determine if an element shall be used.

Transformers Transformers perform static modifications or encoding of its parent elements. Transformers can be used in situations when it is desirable to fuzz an encrypted part of a packet, for example. In such a context the user would define the plain text format with the previously explained elements and then use a transformer to encrypt them.

Fixups Fixups are similar to transformers but instead operate on data retrieved from an another element other than their parent. In other words, while transformers replace other elements, fixups add extra elements. A typical use case are checksums.

3.3.3. STATE MODEL

State models are the elements Peach provides users to manage the flow of data during fuzzing, in other words, to describe the protocol state machine. They are made up of one or more state elements as well as at least one action element.

Action As hinted by its name Action elements perform numerous actions in the parent state model. The set of available actions can vary between Publishers (see Section 3.3.5) but common ones are sending outputs, receiving inputs or changing states. Possible child elements of Action are data models but also sets of data from individual files.

State States group together multiple Actions which are run sequentially.

3.3.4. AGENT

One of the critical steps of fuzzing an application is detecting when error conditions occur. Only after such events it is possible to investigate the root causes and determine whether they are exploitable. The definition of errors in fuzzing is highly dependent on the nature of the target and consequently so are the monitors. Anomalous behaviors range from high memory usage, to increased power consumption, or unusual latencies. Choosing which metrics to oversee plays a vital role in the success of a fuzzer, agents and monitors are used for these purposes. Peach includes a multitude of debuggers and other type of monitors across Windows, Linux and OS X for both local and remote fuzzers.

The Agent is a top level element that aggregates the definition of one or more monitor configurations. Agents are instances that work in tandem with the fuzzer to monitor the behavior of the target. They can can run locally or remotely and perform tasks as distinct as intercepting network traffic, attach debuggers or detect pop up windows.

3.3.5. TEST

Test elements link together the previously defined state models and publishers to configure a unique test case. In addition, test elements are used to exclude elements from being fuzzed, select agents, and choose which mutation strategy to use.

Publishers Publishers are responsible for the I/O communication. Peach includes some Publishers for TCP or HTTP communication but it is also possible to program custom pub-

lishers who implement instances of generic I/O actions such as initialize, finalize, send, receive, etc.

Strategies There is a myriad of strategies built-in Peach that define how the model is mutated. The existing mutation strategies are divided in two main categories: sequential and random.

As indicated by the name, sequential mutations are deterministic and progress from the first to the last fuzzable field. Each field is fuzzed only once. Within this category there is a distinction between linear (`SequentialMutationStrategy`), and random progress (`RandomDeterministicMutationStrategy`) . These approaches are similar with only the order changing, which may allow the fuzzer to detect flaws earlier on the test suite.

On the other hand, random strategies are not finite and fuzz up to N elements per iteration which allows for more flexibility and bigger coverage since combinations of fields are also fuzzed. The downside is the running time. Therefore, a typical test starts with sequential strategies and then proceeds to random ones. For reproducibility purposes, it is also possible to specify a seed for the internal random number generator.

In this category there are three mutators: `RandomMutationStrategy`, `SingleRandomMutationStrategy` and `DoubleRandomMutationStrategy`. The only distinction between the first and the others is the maximum number of elements that are fuzzed per iteration, which is 7 instead of 1 or 2. Nonetheless for `RandomMutationStrategy` the maximum value can be arbitrarily chosen. Furthermore, it is possible for developers to program their own strategies.

3.3.6. RUN

The last top level element available in Peach is the run. At the most essential level, this element ties together one or more test elements into a run. Optionally it can also include a logging procedure to store intermediary information about the run.

If a pit file contains multiple runs, they can be invoked from the command line by passing their name to the peach runtime. The default name for a run is DefaultRun.

4

eFuzz

Available literature [11–16] on smart grid security is largely focused on higher level designs or guidelines that aim to raise awareness, and assist stakeholders implementing this complex network in a relatively secure way since this is a nascent field.

In comparison, few researchers have contributed tools to evaluate devices already in the market. In particular for smart meters, one of the exceptions that is freely available is Termineter [61]. Termineter is a python framework for security testing that supports ANSI C12.18 and C12.19, standards that are commonly used for smart metering in the United States. On the commercial side ProtoPredator for Smart Meters (PP4SM) [62] enables fuzzing of ANSI C12.18 meters through the optical interface.

However in the European market, ANSI C12.18 is not as popular. Instead, DLMS/COSEM is the *de facto* communication protocol for smart metering application. Still, to the best of the author's knowledge, there are no similar applications, commercial or non-commercial.

To address this meaningful void in smart grid security a fuzzer was developed particularly for DLMS/COSEM smart meters. The main purpose of the fuzzer, affectionately called eFuzz, is to improve the security of smart meters, specifically the implementation of DLMS/COSEM. This tool can complement security evaluations of smart meters as well as assist OEM developers prevent unsafe code from reaching the market. To expedite the development, eFuzz uses Peach has the underlying framework.

4.1. Fuzzing requirements

To obtain successful results with fuzzing it is beneficial to follow a set of guidelines and best practices. The summary that follows is based on [59] and adapted to the specificities of eFuzz.

Reproducibility and reusability Reproducibility and reusability are fundamental for a tool that is meant to be expanded and used in different contexts. To uncover security flaws with fuzzing tests, crashes need to be easily reproduced. Peach provides deterministic mutation strategies exactly for this purpose. Moreover it is also possible to specify exactly which test cases should be included in a run, providing complete control to the user.

Reusability is relevant when expanding eFuzz and adapting it for different targets.As will be further explained in the subsequent sections eFuzz is a proof of concept application and thus only covers a subset of the DLMS/COSEM complete specification. Thereafter it is realistic to expect further improvements will increase the coverage of the standard. To accommodate for this requirement the Peach Pit makes frequent use of inheritance to improve readability and encourage code reuse. Furthermore, during the development of eFuzz various custom self-contained modules were written which can be reused without modifications even if the Pit significantly changes.

Process state and process depth State and depth relate to state machine that makes up the target protocol. Virtually all protocols, DLMS/COSEM included, go through a set of stages and transitions that determine their actions. States can be responsible for tasks such as authenticate, commit operation, retrieve data, etc. Furthermore certain states are deeper in the decision tree than others, and can only be reached through a specific path. For example, retrieving data may only be possible after successful authentication. In fuzzing, these paths must be transmitted to the tool, so the generated test cases can penetrate into deep states of the protocol state machine.

In eFuzz this consideration for the protocol state machine is reflected in different elements. As an example some of the protocol fields, such as opening and closing flags are not fuzzed, and both checksums and size fields are computed correctly.

Code coverage Another aspect that is influential to fuzzing quality is code coverage. Code coverage describes the amount of states and transitions a fuzzer can reach and execute. The concept is similar to the aforementioned process depth but it is closely correlated to the underlying code responsible for the implementation of the protocol. In the context of this thesis,

access to the source code of the DLMS/COSEM was not a possibility, therefore it is not possible to measure the coverage provided by eFuzz. Nonetheless for the original developers, a tool like eFuzz, can be optimized for coverage and used in the quality assurance efforts. Flaws detected at such an early stage are less costly to correct, and do not end up in production products where they may pose as a latent vulnerability.

Error detection Detecting errors is as important as causing them. Monitoring of the target in fuzzing is crucial to detect bugs and to collect enough information for further investigations. However monitoring is often target specific and thus not always easily portable. This difficulty is exacerbated when the targets are embedded systems, since collection of metrics is often more challenging than in traditional software applications. eFuzz does not presume any access to internal meter data, therefore it relies on responses to reference queries to ascertain the health of the fuzzed devices.

4.2. COMPONENTS

As described in detail in Section 3.3, Peach is divided in different components according to their task which are all orchestrated by a Pit file. In this context, the most relevant ones are the "transformers", "fixups" and "publishers". Peach comes pre-installed with several instances of each component. However, since embedded systems are not typical targets for fuzzing, it was necessary to develop custom versions to communicate with the meter under test.

Figure 4.1 shows an bird's-eye perspective of the aforementioned components as well as their interactions and dependencies. The Figure makes clear that the Transformer, topic of Section 4.2.1, is responsible for the Galois Counter Mode (GCM) cryptography operations. The same also section includes more details on GCM. Furthermore, the Fixup, discussed in Section 4.2.2 computes the FCS (frame check sequence) checksum. Next, the Publisher implements the serial I/O operations that are used to communicate with the smart meter, via an Infrared optical port. Section 4.2.3 further explains publishers. Finally the Pit, which is examined in Section 4.2.4, coordinates the three components and provides the DLMS/COSEM model to fuzz the target.

4.2.1. TRANSFORMERS

Very succinctly, transformers perform static modifications or encoding of the parent elements. They can also behave bidirectionally, *i.e.* encoding and decoding. In eFuzz, a custom

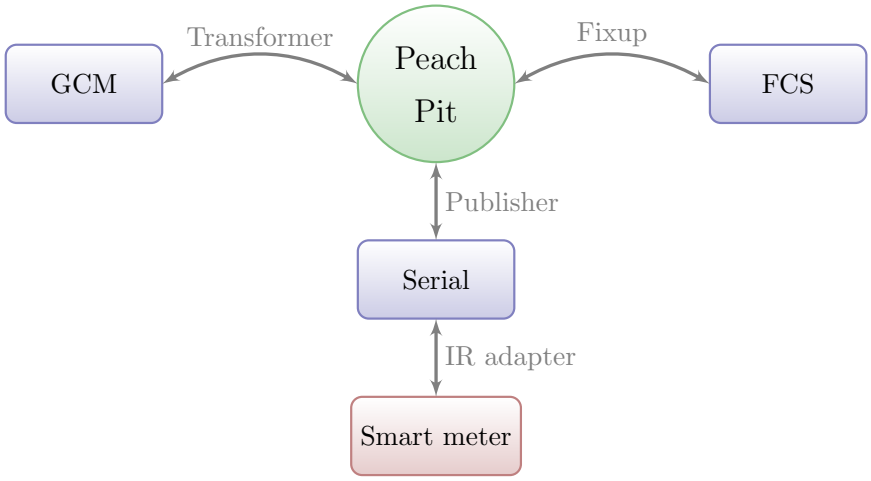


Figure 4.1: Different components used in eFuzz/Peach and their relationship.

transformer was written to encrypt and decrypt the APDU payloads.

The CryptoTransformer implements the Transformer class. The class consists of two functions and the constructor.

To allow for easy portability eFuzz includes a configuration file that allows user to quickly add support for new devices or edit the existing ones without having a deep understanding of Peach or DLMS/COSEM itself.

The constructor reads the configuration file, opens it, and reads the fields it needs to encrypt and decrypt the APDUs. The first is the `CRYPTO_KEY`, the cryptography key for GCM. The second, `AUTH_KEY`, corresponds to the authorization key used for authentication. This key is used to generate the tag that verifies the authenticity of the sending party, be it the server or the client. The third and fourth fields are `PC_SYSTEM_TITLE` and `METER_SYSTEM_TITLE`. These are public values that identify the client and the server, respectively. Both strings are used as part of the initialization vectors along with a frame counter to avoid replay attacks.

The first function `RealEncode` transform the plain text requests into the encrypted equivalent. The function returns this representation.

The second function `RealDecode` does the opposite, after receiving a request from the meter (server) decodes it so it can be interpreted by eFuzz. The return value is a hex string of the plain text version of the original APDU.

As explained, the transformer is used to encrypt the mutated plain text to the final form. In the current context, all necessary keys and passwords to encrypt and decrypt the text were

available. Presuming access to this data can be acceptable or not depending on the context. For an "ordinary" attacker, it is not a realistic premise. Moreover, since every meter should be deployed with its own keys, in case of key disclosure only one device would be compromised. However, in the context of a security evaluation this is an acceptable assumption. For example, a company hired to assess the security of the device would be given access to the keys and perhaps the firmware source code. Then, they would use eFuzz in an initial approach to autonomously detect possible points of interest. The respective report would serve as a guide to security experts and complement a full featured source code review. Moreover smart meter manufactures can also greatly benefit from eFuzz by integrating it in their test suites, thus detecting flaws early in the product development process.

GALOIS COUNTER MODE

The Galois Counter Mode (GCM) of operation combined with the Advanced Encryption System (AES) block cipher forms the basis of encryption and authentication in DLMS/COSEM. Due to its importance, this section provides a summary of AES-GCM properties and underlying algorithm, based on the information provided in [3].

GCM considers two operations: authenticated encryption and authenticated decryption. Figures 4.2 and 4.3 show examples of authenticated encryption and authenticated decryption in GCM, respectively.

The first operation, authenticated encryption, receives four inputs:

1. A secret key, K, with the appropriate length for the block cipher. In DLMS/COSEM GCM is used with AES-128, hence K should be 128 bits long.

2. An initialization vector IV, with an arbitrary length between 1 and 264 bits. For efficiency reasons, the authors recommend the use 96-bit vectors. DLMS/COSEM abides by this recommendation. This vector is generated by the party that is responsible for computing the encrypted authentication. It is important to note that using the same IV with two different messages encrypted by the same key voids the security properties [63].

3. A plain text, P, between 0 and 256 bits.

4. Additional authenticated data (AAD), A. This value is authenticated but not encrypted, thus is not included in the output. When using authenticated encryption in DLMS/-COSEM, A is the concatenation of the security control and the authentication key. The

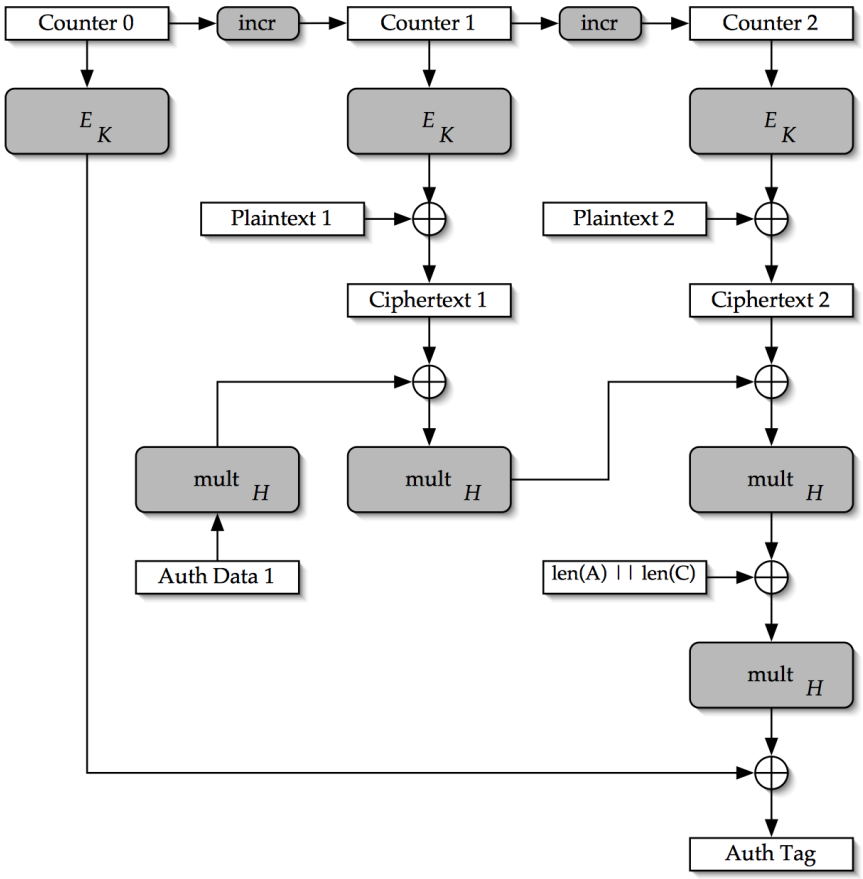


Figure 4.2: The authenticated decryption operation. This Figure depicts a simple case with only a single block of additional authenticated data (labeled Auth Data 1) and two blocks of plain text. E_K denotes the AES encryption using the key K, mult_H stands for the multiplication in $GF(2^{128})$ by the hash key H, and incr designates the counter increment function. This Figure is originally from [3].

security control is a byte long value that indicates the level of security to use.

And produces two outputs:

1. A cipher text, C.
2. An authentication tag, T, with length between 0 and 128 bits. In DLMS/COSEM, the length is set at 96 bits.

Authenticated decryption has similar inputs: K, IV , C, A, and T. And only one input, either the plain text P, or a FAIL symbol that indicates the authentication process did not succeed.

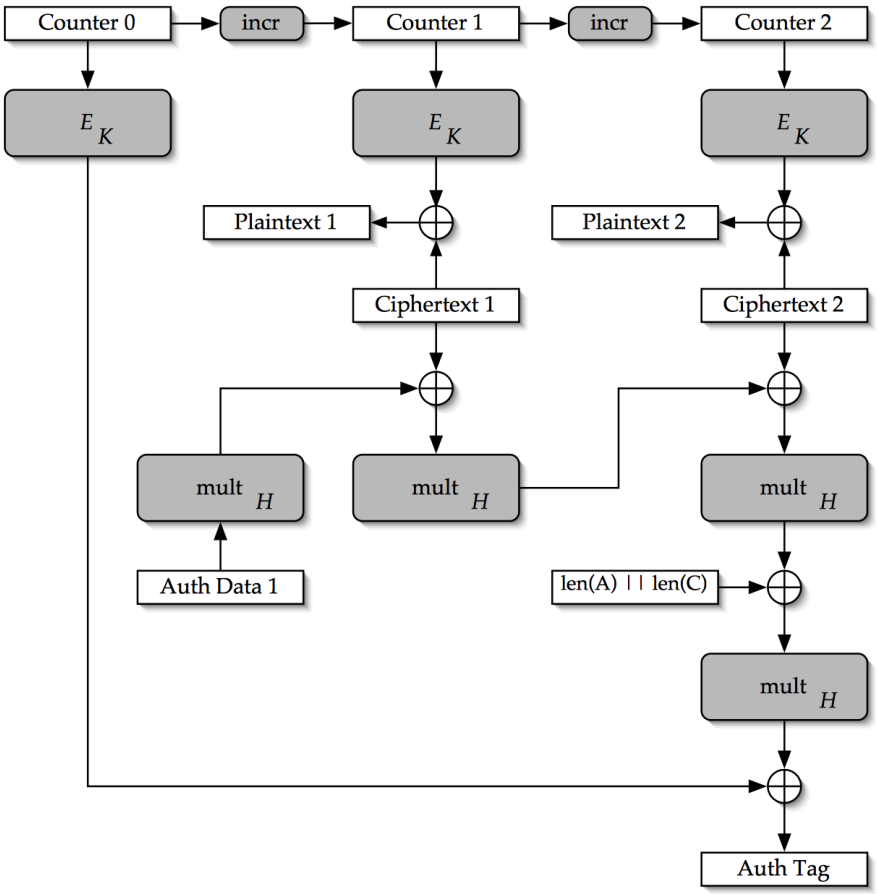


Figure 4.3: The authenticated decryption operation, considering the same case as in Figure 4.2. This Figure is originally from [3].

According to [64], GCM provides assurances with respect to confidentiality of the data, and authenticity of confidential or non-confidential data using the aforementioned hash function H, defined over a Galois field. Furthermore GCM can detect both accidental modifications of data as well as intentional, unauthorized alterations. With respect to performance, GCM is very efficient and can be run in parallel. Consequently, high-throughput implementations for both software and hardware are possible.

The AES-GCM variant used in DLMS/COSEM, combines AES counter mode for encryption, and the GHASH algorithm for authentication. GHASH is defined in Equation 4.1.

$$\text{GHASH}(H, A, C) = X_{m+n+1} \tag{4.1}$$

In Equation 4.1, H represents the is the Hash Key, encrypted using the block cipher (AES). A and C are the same as explained above. m and n are the total number of 128 bit blocks in A and C, respectively. Note that the last block of A or C, can be smaller than 128 bits. Finally, X_i for i between 0 and $m+n+1$ is defined as follows:

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & \text{for } i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (A_m^* \| 0^{128-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_i) \cdot H & \text{for } i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_m^* \| 0^{128-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\text{len}(A) \| \text{len}(C))) \cdot H & \text{for } i = m+n+1 \end{cases} \tag{4.2}$$

In Equation 4.2 v is the length, in bits, of the final block of A and u is the corresponding length over C.

4.2.2. Fixups

Fixups, which are similar to transformers, operate on data retrieved from other elements to generate new values. In other words, fixups are used to produce values algorithmically, based on data from other fields. Therefore fixups are ideal for checksums. In fact one of the built-in Peach fixups is dedicate to CRC32.

In eFuzz a fixup is used to determine the Header Frame check sequence (HFS) or the Frame check sequence (FCS). Both HCS and FCS use the same code but are computed over a different set of data.

Just like the publisher, it is a modular implementation based on the abstract Fixup class. The only function that is required to be implemented in Fixup instances is fixup. A reference implementation in C of the algorithm can be found in [65].

In eFuzz this function simply calls the C-to-Python translation of the reference implementation, with the content of the corresponding APDU. The APDU fields are passed on the class constructor which unpacks the arguments and makes them available for the rest of the class, in this case the only method that it includes.

As return value the Fixup functions outputs the four-byte sequence that makes up the FCS or HCS.

4.2.3. PUBLISHERS

Finally, publishers are responsible for the I/O communication, in this case serial. eFuzz uses the pySerial [66] library to enable cross-platform serial communication. In addition the infrared port, the meter used as target is equipped with a GSM radio. However since the latter relies on a complex infrastructure outside the control of the author, it was deemed advantageous to use the optical port. The physical connection between the meter and the laptop running eFuzz was accomplished with an appropriate ANSI type-2 optical probe with a serial USB interface.

More publishers can be added for different connections without any need for changes in other parts of the code. The abstract publisher class defines a set of functions that must be implemented by the different instances. One of the functions, initialize, which opens the session, is called at the start of each test run. The counterpart, finalize closes the session. In addition, the send function publishes arbitrary data, passed as a function argument, to the open port. And finally, received reads and returns the received data. This function can optionally be invoked with the expected number of bytes.

INITIALIZATION

When the serial publisher is first started it assumes the following settings for the serial connection:

- Baud rate - 300 bits per second
- Byte size - Seven bits
- Parity - Even parity
- Stop bits - One stop bit

These settings are defined in the specification and were introduced in Section 3.1.4.

Since the target sometimes does not reply to requests it is important to specify timeouts. These values were determined experimentally as a good balance between speed (to avoid waiting too long when the meter does not response) and reliability (to avoid false errors).

All serial communication is retrieved byte by byte regardless of the total size of the message. This aspect allows eFuzz to always receive the complete message regardless of the size. Not trusting the size announced by the meter adds to the robustness of the program, as the meter may be malfunctioning.

For receiving a response the timeout is set at 0.2 seconds. There is no timeout to write a byte to the port (*i.e.* the program waits until the bytes are written to the port). Eliminating the write timeout allows slower machines to run the fuzzer, which could otherwise trigger exceptions.

eFuzz uses a common function to read content from the serial. If the size is known beforehand it tries to read that size, however if it fails to acquire that specific amount of bytes after multiple attempts it signals an error. Alternatively, if the size is unknown (which is the case every time a fuzzed request is sent) it will try to read as many bytes as possible.

Due to the context of this project it is crucial to have a robust program, in particular, as the target may not always respond as expected or may take more time than anticipated. In addition, the software is meant to be used in different configurations, which reinforces the same argument.

Direct communication follows a set of procedures, as shown in Figure 4.4. This diagram was originally shown in Section 3.1.4, and is repeated here to aid the reader.

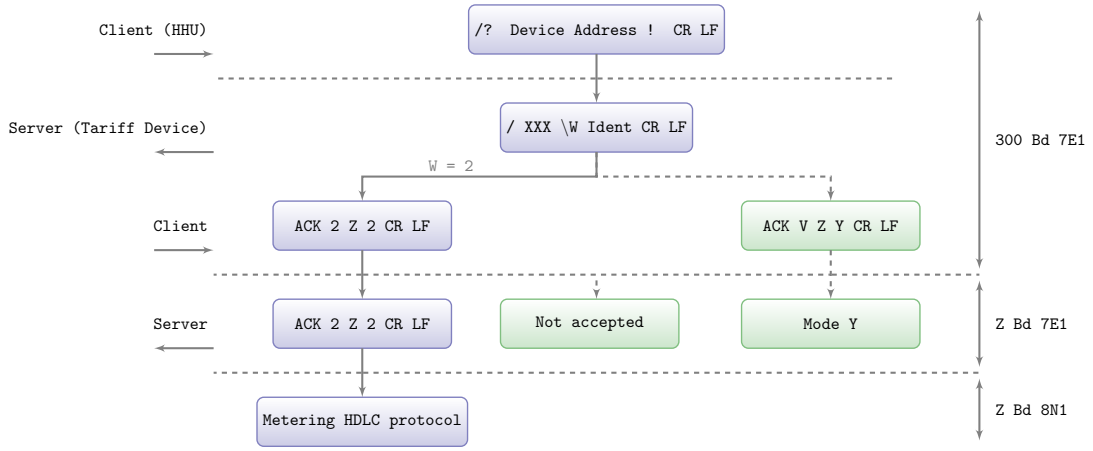


Figure 4.4: Entering protocol mode E (HDLC)

The first step of direct communication is to send the 1st handshake, which the server should acknowledge with the message shown in Figure 4.4.

However if the response does not match the expected string, eFuzz will try again after 1 second. If after that time, the response is still invalid, the process repeats itself with an exponentially growing waiting time. After reaching 32 seconds, the waiting time no longer increases.

The following message exchanges to enable normal mode and define HDLC parameters, with SNRM messages. This follows closely the process with the exponential back off.

To establish the session, only one step remains: the AARQ and AARE. Until this point all the messages were sent unencrypted. This is the first exchange that takes into account the security policy, and the respective encryption and authentication keys.

If all three steps are successfully executed, the initialization process is complete.

DLMS APDUs

After the initialization phase it is possible to send APDUs to the target. The publisher implements the function used to send the bytes with the correct serial port configurations. Assembling the actual request is done in the Pit file.

TERMINATING OR RESTARTING THE SESSION

To close the session there is also a specific procedure, based on the DISC type frame, which the publisher implements.

When eFuzz requests the publisher to terminate the session, the publisher simply sends this message and disconnects by closing the serial port.

Alternately, the publisher can be instructed to restart the session. This occurs when the meter has reached a state where it is not responding properly. Such behavior is signaled to the publisher which sends the disconnect message followed by the initial handshake, described in section 4.2.3.

4.2.4. PIT

The Pit brings everything together to start the fuzz testing. As explained in section 3.3 one of the most important elements of Peach Pit files are Data Models. eFuzz uses several of these data abstraction layers to construct its APDUs and it does so in a way that allows for great flexibility to swap components. The most general data mode is called `hdlc-wrapper` as it provides the necessary data for the transport and data link layers. The remaining four Data Models all inherit `hdlc-wrapper`.

eFuzz uses a modular approach to generate the fuzzed requests. As mentioned, the most abstract data model is a common wrapper. The block incorporates the HLDC and the LLC layers. Some of the fields are randomly chosen while others, such as size and checksums, are computed deterministically. The checksum, also known as frame check sequence, is handled by the custom fixup instance described in Section 4.2.2. The DLMS payload is declared as

well, but the actual contents are not specified.

There are two distinct models describing the APDUs sent from the PC to the meter. The first APDU inherits the common wrapper and only replaces the necessary fields. The request is fuzzed at the plain text level and is thereafter encrypted, through a user-defined transformer. This APDU is modeled after a request that queries the meter for the local time and date. The second APDU defined by eFuzz is only used as a reference. It is similar to the previous one but none of the fields are modified.

The two APDUs that correspond to the answers from the meter serve different purposes. The response to the fuzzed request is read but disregarded. The sole purpose of the first APDU is to provoke faults in the target therefore the response has little relevance. The following response is used to assess the health of the meter. Since it corresponds to a reference request can be compared with a reference response. If the received and expected data do not match, eFuzz logs the event.

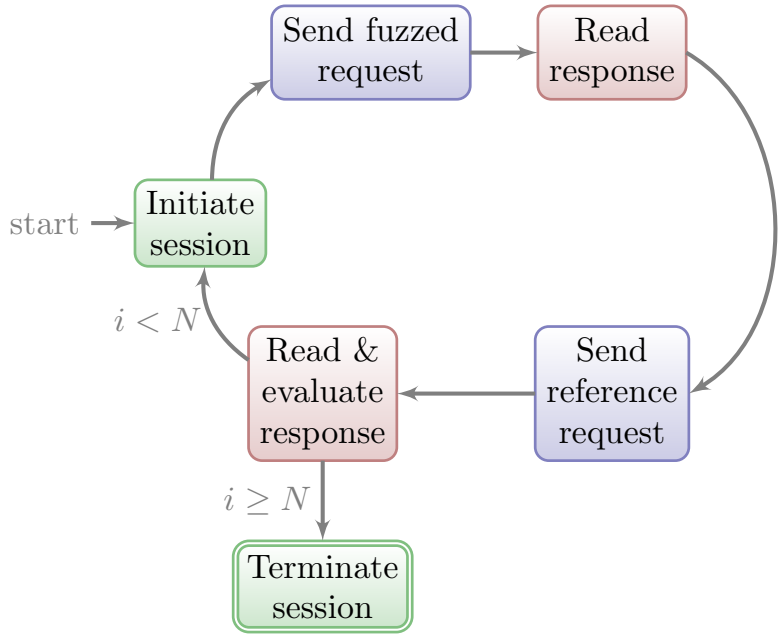
4.2.5. State Machine

The response of the meter to the invalid queries is not necessarily relevant as there is not correct or incorrect response. More interesting is observing how they might affect the meter behavior thereafter. Given the specificities of DLMS/COSEM two main ways were chosen to do so (see Figure 4.5).

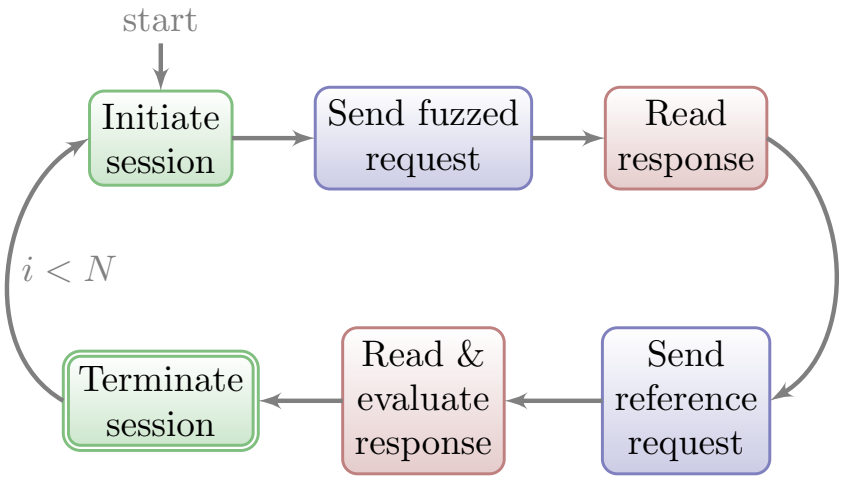
The first approach is to send a reference request after each fuzzed packet. If the response matches what is expected, the meter is functioning properly. Otherwise, this behavior should be signaled and stored for human inspection. Thereafter the process is repeated. An exception to this cycle occurs when the target does not respond as expected. In this situation the session is restarted. Otherwise, the subsequent tests would not be evaluated by the meter as it was already in a faulty situation. As an example, assume request i causes the meter to reach an unknown state that rejects all further communication in the session. Then even if request $i+1$ would normally trigger a new problem it would go unnoticed. The state machine describing this behavior is show in Figure 4.5a.

The second approach is similar but the session is restarted every time a fuzzing attempt terminates, as portrayed in Figure 4.5b.

These contrasting approaches may yield different results. The former is expected to detect more memory leaks or other memory related problems. For example, a memory leak triggered by an attack may not be immediately noticeable and only have consequences later



(a) Fuzzed and reference packets are all sent within the same session.



(b) The session is restarted after each fuzzing attempt.

Figure 4.5: State machines describing the two approaches used to fuzz the target device. On these diagrams i indicates the current fuzzing iteration while N is the total number of tests available. In both situations the first iteration uses a reference request to ensure the target is in a known state.

on. If the session was terminated in between, there would be a chance for the meter to clear its state and recover from such a problem.

The state machines are modeled in Peach using StateModel elements. eFuzz defines two

of such elements, one corresponding to the scenario described in Figure 4.5a and another corresponding to Figure 4.5b.

4.2.6. Tests and Runs

Test elements combine the state models and the publisher. Furthermore in eFuzz they prevent the reference Data Models from being fuzzed. eFuzz defines six instances of Test elements, in order to integrate different mutation strategies and the pair of state machines available.

Within the six tests, there are three for each state machine. The difference between this three pertains to the mutation strategies. One uses a RandomDeterministicMutationStrategy, another a DoubleRandomMutationStrategy, and the last uses a RandomMutationStrategy with up to seven fields mutated at a time. The differences between these strategies are outline in Section 3.3.5.

The last top level element in eFuzz's Pit file are two Run instances. Again each run correspond to one state machine. These elements are used to invoke the just described respective tests. Moreover logging configurations, important for debugging and post fuzzing investigations, are also defined in these elements.

4.3. Post processing

eFuzz gathers data about all the test cases it executes, which results in rather long log files. Therefore an auxiliary routine, named Reporter, was written to filter such files and present only the situations where the target behave unexpectedly. The user can then use this file as main reference, accessing the original complete traffic if necessary.

To detect whether a certain message exchange is valid or not the Reporter looks for a correctly decrypted APDU in the reference response for each sequence of four messages. The sequence corresponds to a complete state machine cycle, *i.e.* fuzzed request, fuzzed response, reference request, reference response. If such strings can not be found then the meter responded with a different message which is then copied to the error report.

The resulting report can be used by an expert to assist in the post-fuzzing operations. Depending on the specific context, there are various possibilities for what to do with the hypothetical eFuzz findings. One option is to report the bug in its current state, for which case no more effort from the user of the fuzzer is required. Alternatively, the user go one step further and try to determine exactly what situation triggered the error. This requires further

investigation and experimentation to pinpoint the exact cause. If the source code is available, it might be fruitful to inspect the corresponding code, as well. After identifying the root cause and determining it is exploitable, the following phase is to actually write the exploit.

The process of writing an exploit is intrinsically linked to the target and its specificities, even more for embedded systems. Nonetheless, it typically relies on manipulating memory values. At this point, it would probably be necessary to open the meter and try to gain access to such low level memory information in order to proceed.

The readers interested in this process are referred to [67] which details how to develop an exploit based on a vulnerability discovered through fuzzing with Peach.

5

Results & Analysis

To assess the performance of eFuzz, this Chapter presents the results obtained during various experimental procedures. Although the main goal of eFuzz is to provide a versatile and easily expandable application to fuzz DLMS/COSEM smart meters, it is, nonetheless, illustrative to study preliminary results obtained during the development of said tool.

In general, one of the major goals of fuzzing is to provoke crashes or some other type of malfunction in the target. In traditional software applications, where this technique originated, detecting anomalous behavior is relatively straightforward. For example, if the target is a web browser several metrics can be easily monitored. For example, applications such as WinDbg, a post-mortem debugger for Windows, can be used. Memory consumption and CPU usage are other variables that can be observed to indicate if the target has been affected by the fuzz testing. Furthermore, a Peach agent can be configured to kill and restart the process if certain conditions are met. This is particularly beneficial for long running fuzzing sessions, since it makes the tests completely autonomous.

In embedded systems, such as smart meters, those metrics are harder to observe. One reason is that embedded systems are a much more heterogeneous class of devices. While in software applications there is a shared operating system layer that the fuzzer can inquire to monitor the target, for embedded devices the best way to handle target errors varies widely. Certain devices possess JTAG ports for debugging purposes which can be used for monitoring purposes. In other situations using optical sensors to monitor the status of certain LEDs is

the preferred method. Alternatively, studying the latency of the target response may be the most fitting procedure.

In eFuzz, like in any other fuzzer, which aspects to monitor is crucial to its success. Based on the requirements explained in Section 4.1, the monitoring technique has to respect a few restrictions:

1. No physical modification of the target. For the identified attack scenarios it is not reasonable to assume access to the meter internals. In particular, home owners, are not likely to violate the meter seals as doing so becomes immediately evident and may lead to direct penalizations. Furthermore physical modifications require personal access to the meter which excludes remote attack vectors and are not scalable.

2. The monitored variable must be persistent across devices. Given the goals of eFuzz that pertain to portability it is crucial that the chosen metric is available in all meters, regardless of the communication interface it is used.

3. It must be simple to determine success or failure. In other words, it must be evident and unambiguous whether the target is working properly or not. Imposing this rule may be too restrictive with respect to the monitors available. However, for a first approach to meter fuzzing, like eFuzz, it is advantageous to have a simple test to check the health of the target.

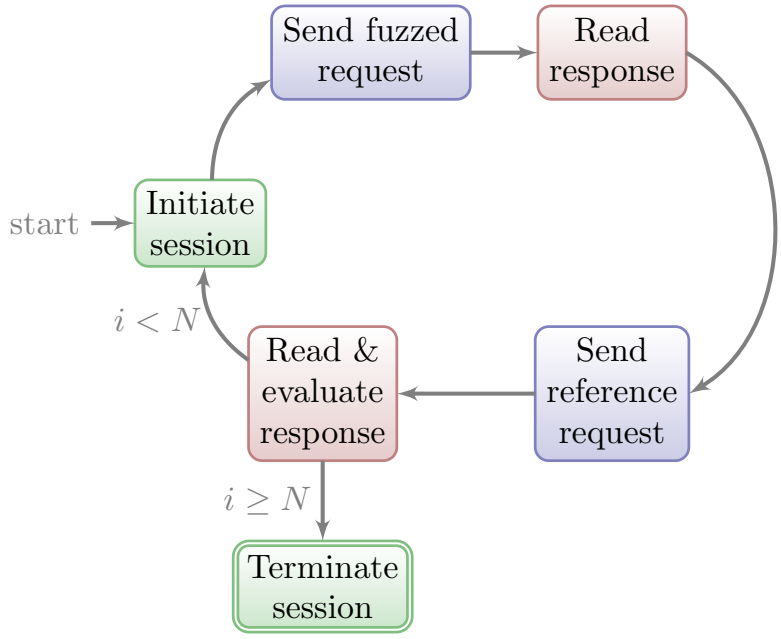
Given the outlined selection criteria, it became evident that using a reference query, reference response message exchange would be an appropriate choice. It does not require physical access or alteration of the target since it only consists of protocol message exchange. In addition, it is not specific to the current target. Instead, it is part of the DLMS/COSEM specification and thus is present across all meters that implement the standard. And finally it is trivial to check for success or failure, either the response matches the reference or it does not.

The reference request that is used is requesting the meter's date and time. This can be retrieved with a simple, and short, dialog with the target that can be easily checked. Therefore, the results, shown in Section 5.1, reflect the number of times the meter did not respond correctly to the reference query.

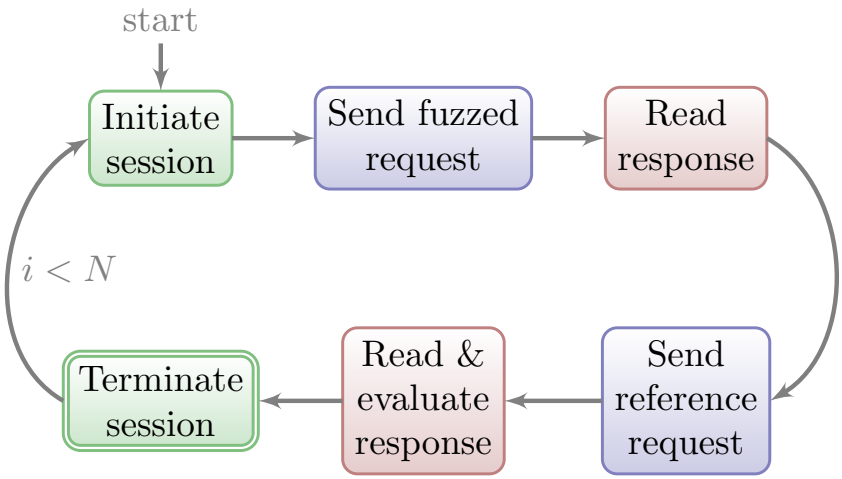
5.1. DETECTION PERFORMANCE OF EFUZZ

This Section presents the amount of unexpected behaviors provoked in the target. The fuzzing procedure follows the state machines shown in Figure 5.1. The Figure was originally featured

in Section 4.2.5 but it is repeated here for convenience.



(a) Fuzzed and reference packets are all sent within the same session.



(b) The session is restarted after each fuzzing attempt.

Figure 5.1: State machines describing the two approaches used to fuzz the target device. On these diagrams i indicates the current fuzzing iteration while N is the total number of tests available. In both situations the first iteration uses a reference request to ensure the target is in a known state.

As shown, after sending each fuzzed request, eFuzz sends a reference query and compares the result with the correct response. If these values do not match, the dialog is stored as an

unexpected behavior and the session is restarted before continuing. The reference request consists of asking the meter for its internal time and date.

5.1.1. RESULTS

Due to the detail used to model the protocol, the complete test suite of eFuzz when using the sequential strategies includes 54727 distinct cases. Running all tests requires around 7 days. However to be able to obtain data for different configurations, within a reasonable time frame, a subset of 500 tests was used. The subset completes within approximately 3 hours.

Table 5.1 shows the number of unexpected behaviors induced with three distinct strategies. Sequential Random refers to the aforementioned `RandomDeterministicMutationStrategy` class. The other two use random strategies, where n specifies the maximum number of fields that are fuzzed per iteration.

Mutation strategy	**Number of unexpected responses**
Sequential Random	40
Random $n = 2$	21
Random $n = 7$	10

Table 5.1: Overview of the number of unexpected responses caused by eFuzz discriminated by strategy. Variable n indicates the maximum number of fields that are mutated in each iteration. In the sequential strategy this value is by definition one.

The amount of invalid responses is significant, in particular considering the short amount of time it required. The sequential strategy was the most fruitful, reinforcing the point made above that this strategy should be used first.

With respect to the random strategies there are several possible explanations for the lower number of unexpected responses. First, in these runs Peach includes iterations where it does not mutate any of the fields. Therefore it sends two reference requests consecutively, which results in a “wasted” iteration. A second reason, is that by increasing the number of invalid APDU fields per request, it is more likely that at least one of the boundary checks included in the code is triggered, causing the meter to ignore the request. This also helps explain why the increase in n results in less invalid responses.

In addition to the quantity, it is also insightful to understand what the meter response actually was. This information is condensed in Table 5.2. The responses are separated by

Mutation Strategy	Exception-response	Frame Reject	Other
Sequential Random	50%	50%	0%
Random $n = 2$	57%	33%	10%
Random $n = 7$	50%	40%	10%

Table 5.2: Distribution of the unexpected meter responses by cause.

exception-response, frame reject (specific DLMS/COSEM messages) or other.

The Exception-Response APDU is an optional APDU sent by the COSEM server application layer that indicates to the client application layer the service requested could not be processed. Particularly for the case shown, the request was not processed as access to the service was not allowed. Alternatively the server can simply discard the message without sending this APDU.

The Frame Reject (FRMR) is used to report one of various conditions occurred and retransmission of the identical frame can not correct it. The situations that may trigger an FRMR include the receipt of an undefined or not implemented command or response, receipt of a frame that is too long, receipt of an invalid frame, or receipt of a frame containing a field that is not allowed.

Categorizing the meter responses shows a clear majority of `exception-response` and `frame-reject` APDUs. These messages are used to signal errors and should not occur in response to valid requests, which suggests one of the previous messages perturbed the internal state of the server and has caused undesirable side effects.

5.2. DISCUSSION

DLMS/COSEM is a complex protocol, and consequently so is its implementation. In manual evaluations such as source code reviews, it is very costly to inspect the complete code. Furthermore, considering all possible combinations of inputs that lead to different execution paths is virtually impossible for humans, due to its sheer complexity. In comparison, fuzzing is an appropriate technique for such operations. Fuzzing is completely automated and often consists of thousands of tests cases, providing excellent coverage.

On the other hand, fuzzing is not a security panacea. As explained in Section 3.2, fuzzing is not capable of detecting certain flaws with security implications, such as check for correct user permissions, detect poor design logic, discovering back doors, identify memory corrup-

tion (unless it leads to crashes) or expose multistage vulnerabilities. In fact, all of these issues are more likely to be exposed by human inspections.

The results presented in Section 5.1 indicate this preliminary work exposes bugs that would probably go unnoticed otherwise. Just like in traditional software applications, fuzzing is a good technique to test proprietary code. The approach used can generate combinations of inputs that would not be possible by a human tester, as she would be more inclined to follow the same assumptions as the original developers.

A series of experiments performed on the target were able to provoke between 10 and 40 incorrect responses, depending on the mutation strategy selected, in a short period of time. Although with this approach it is not possible to automatically determine if these bugs have security consequences, it pinpoints areas of interest that a human expert can focus on. Despite the high bug count it is expected that the underlying code has a lower number. The overestimation happens because the same bugs are very likely triggered more than once by the fuzzer, in different test cases. Nevertheless, it is remarkable that eFuzz was able to cause mischief in a commercial product, despite the short time allocated for testing.

Regardless of its success, this method is not meant to replace professional human auditors, or OEM developers, but instead complement them. The added value is accomplished by means of an automated tool that exhaustively finds bugs with respect to diverse input combinations. Nonetheless, note that manual inspection by a human expert cannot achieve the same level of testing within an acceptable time frame.

To better understand the complementarity of both approaches, the results of eFuzz and an actual (manual) security evaluation are compared. Both studies used exactly the same smart meter model. Table 5.3 compares the two with respect to four different parameters.

One of the main advantages of fuzzing is its automated nature. Automation makes it faster, hours or days instead of weeks, than manual evaluations and requires little human supervision. On the other hand eFuzz has a very specific coverage while a manual approach can be much broader and detect classes of vulnerabilities that a fuzzer can not, for example if keys are stored securely or if permissions are properly enforced. As previously shown eFuzz is able to detect a large quantity of findings while manual inspections are typically more selective. This disparity is reflected in the quality of each discovery. Issues uncovered by fuzzing need to be inspected by an expert to determine if they constitute a real security threat.

Parameter	eFuzz	Manual
Time	Hours/few days	Weeks
Coverage	DLMS/COSEM logical attacks	Physical attacks. Weaknesses in configurations, key storage, retrieval.
Quantity findings	High	Low
Quality findings	Low	High

Table 5.3: Comparison of automated and non-automated approaches to detecting smart meter vulnerabilities.

6

CONCLUSIONS

6.1. SUMMARY

The normal functioning of society depends heavily on the availability of certain resources. Some of the most essential ones are delivered through large and complex infrastructures, often referred to as critical, due to their vitality. One of the most fundamental resources to the economic tissue and individual's life is energy, in particular electricity.

The electricity grid is of particular importance, due to the structural evolution it is currently undergoing. The investments in the grid will provide networking capabilities to all its constituents, as described in Chapter 3. For example, the ability to remotely monitor and control all grid activities will become a reality.

With the advent of ubiquitous communication competences in smart grid devices, such as smart meters, a large emphasis has been directed attention to the security of such devices. Due to the new abilities, numerous adversaries that have paid little attention to devices such as smart meters, have now reason to study them. In particular, with the ongoing escalation of cyberwarfare activities, the narrative to secure critical infrastructures has gained a new sense of urgency.

One of the most crucial components in the smart grid are the smart meters. In Europe, these devices predominantly rely on the DLMS/COSEM protocol to communicate.

The Device Language Message Specification (DLMS) consists of a general concept for abstract modeling of communication entities. On top of this, the *COmpanion Specification for*

Energy Metering (COSEM) provides a set of standards that define the rules for data exchange between smart grid devices, such as an energy meter and a data accumulator. Together they provide several features, such as (a) an object model to view and access the different functionalities of a meter, (b) an identification system for all data, (c) a method for communicating with the model and (d) a transport layer to accommodate the information flows between the meter and other devices. The protocol is based on the *Open System Interconnection* (OSI) seven-layer model. However in the case of smart meters they are practically collapsed in four - physical, data link, transport and application layers.

DLMS/COSEM is complex protocol and that complexity is reflected in its implementation. Two consequences of writing complex code are, it is likely to have bugs (some with security implications) and it is difficult to manually review it.

One technique that is widely used to find vulnerabilities in communication protocols is fuzzing. It is an automated operations that constantly feeds the target, in this case a smart meter, with malformed data to induce malfunctions. To detect these erroneous actions, fuzzing includes monitoring activities like memory consumption inspections, thrown exceptions, crashes or other unexpected behaviors. Due to its automated nature, it can cover a much wider gamut of inputs, mitigating the difficulties associated with the complexity of the underlying code base.

This thesis proposes a fuzzer, based on the Peach fuzzing framework, for DLMS/COSEM electricity meters, with the goal of testing the security of said devices, specifically their implementation of the protocol.

It was observed that using Peach as the base for eFuzz was a beneficial decision as it reduced the development time. Nonetheless Peach was designed for software applications and most built-in features reflect that. For example the ability to attach a debugger or other types of real-time monitors to the target is not trivial in embedded systems but it is the principal way for Peach to detect successful attacks. Moreover most of pre-installed components, such as publishers, do not include typical features required to interact with embedded systems, for example serial communication. Due to this limitation, users interested in testing these systems need to program the components themselves, which makes the process more arduous.

The fuzzer, described in Chapter 4, works autonomously by generating invalid or malformed DLMS/COSEM packets derived from a model of the standard. By constantly sending these requests to the meter followed by a reference query, one can assess if the fuzzed data was able to interfere with the normal behavior of the target.

In the preliminary tests with a electricity meter currently in the market using a direct optical interface, eFuzz was able to perturb the correct functioning of the target. In less than 3 hours, the tests revealed between 10 and 40 issues depending on the mutation strategy. The majority of the invalid responses indicated that the client was trying to access a service that was not allowed or that the request was malformed, which suggests previous messages caused the meter to reach an invalid state.

Comparing the aforementioned results with the traditional techniques showed that the approaches are complementary. The security evaluation performed by security experts uncovered weaknesses related to the unencrypted storage of keys, proper enforcement of user permissions, among others which are not detectable with fuzzing efforts. However, it did not report any vulnerabilities in the code responsible for implementation of DLMS/COSEM, area where eFuzz was able to provoke multiple errors.

The obtained results show that fuzzing approach is an efficient way to detect flaws in the protocol implementation given the limited human effort it requires. Additionally, the proposed application for fuzzing embedded systems, eFuzz, provides researchers the capability to apply fuzzing techniques as a vital step for the evaluation of smart meters and other networked embedded systems. Ultimately, eFuzz will hopefully make smart meter communications harder to penetrate, and consequently elevate the bar for the security of critical infrastructures.

6.2. CONTRIBUTIONS

This thesis contributes eFuzz, a fuzzing tool that includes a model of a subset of DLMS/COSEM as well as various custom modules to ensure all communications, checksums and cryptography operations are computed properly.

eFuzz can be used by security experts to assist in the evaluation of smart meters that implement this specific communication protocol. It provides a low cost method to uncover potential vulnerabilities in the target's implementation of the complex standard. In addition, it can also prove beneficial for OEM developers as a quality control aid. By incorporating eFuzz in the testing procedures, it can uncover bugs long before they reach their customers. At this initial phase, correcting errors has a marginal cost.

In the introductory Chapter 1, three research questions were introduced as the guiding principles for the thesis work. The answers to these questions are implicit in the content of the previous chapters, notably 4 and 5. Nonetheless, it is worthwhile to formally and directly

address them, which will be the purpose of the following paragraphs.

Research Question 1. *Is it feasible to develop an automated tool with the goal of aiding security assessments of DLMS/COSEM meters, with sufficient flexibility to be easily adapted and expanded to different devices?*

eFuzz shows that efforts to develop an automated flexible tool are indeed feasible. Due to the limited hardware available it was not possible to test the fuzzer across various smart meters. However, test runs were successfully performed in two meters of the same model, but with different configurations.

Research Question 2. *Can such a tool be capable of uncovering vulnerabilities in said devices, within a time frame that is competitive with the conventional approach?*

Preliminary experiments on a commercially deployed electricity meter have shown that eFuzz is capable of inducing errors. Moreover, three hours of testing were sufficient, to provoke up to forty error instances. The promising results show that an automated tool can, in fact, uncover vulnerabilities within a very short time frame. Comparing the results with a non-automated security evaluation further reinforced the competitiveness of the proposed approach.

Research Question 3. *In comparison with manual approaches, are the sets of vulnerabilities uncovered be each, largely overlapping or largely disjoint?*

An interesting conclusion drawn from Chapter 5 is that fuzzing and human security evaluations expose different classes of bugs. Therefore tools such as eFuzz should not be used as replacement to manual efforts, instead they are indicated to assist said efforts amplify the quality and breath of the security experts' findings. In summary, comparing the classes of vulnerabilities uncovered by manual and automated approaches results in mainly disjoint sets.

6.3. FUTURE WORK

As alluded to previously, there is ample room for improvements in eFuzz.

In its current version only a small part of the DLMS/COSEM is modeled, thus one of the most obvious improvements is to expand the coverage of the protocol and consequently of the underlying code.

Another improvement left for future research is considering more nuanced parameters to detect erroneous meter behaviors. Currently a binary comparison is used to check the health of the meter. Considering other variables, such as the latency of the responses, the power consumption, would improve the quality of the monitoring. Yet another possibility would be to open the device and inspect signals closer to the processing units. The downside would be a loss in portability as different meters are likely to have different internal architectures.

An additional possible advancement is exploring alternative communication interfaces, notably GSM, since not all meters are equipped with an optical port. This capability would permit fuzzing through different physical layers and would also enable hybrid test suites. For example, use a direct physical connection to send the fuzzed requests, and the GSM connection to assess the well-being of the target.

At this time, the main fuzzers readily available are not optimized for embedded systems. As networking capabilities are introduced to a growing number of micro-controller based devices, it is fundamental to ensure security is preserved. Fuzzing can play a crucial role in this task, but first it needs to expand its capabilities to these new classes of targets. Therefore, the last suggestion is to investigate improvements that may allow better fuzzing of embedded systems.

A

PAPER SUBMITTED TO SEGS 2014

In the context of this thesis, a paper was submitted to SEGS 2014: Smart Energy Grid Security Workshop, and is here reproduced. The document was written together with Zekeriya Erkin and Christian Doerr from Delft University of Technology, Raymond Hallie from the European Network for Cyber Security and Gerrit van der Bij from Riscure.

eFuzz: A fuzzer for DLMS/COSEM electricity meters

Henrique Dantas, Zekeriya Erkin, and Christian Doerr
Cyber Security Group, Depart. of Intelligent Systems
Delft University of Technology
Mekelweg 4, 2628 CD, Delft, Netherlands
hndantas@gmail.com,{z.erkin, c.doerr}@tudelft.nl

Raymond Hallie
European Network for Cyber Security (ENCS)
The Hague, Netherlands
raymond.hallie@encs.eu

Gerrit van der Bij
Riscure BV
Delft, Netherlands
vanderbij@riscure.com

ABSTRACT

Smart grids enable new functionalities like remote and micro management and consequently, provide increased efficiency, easy management and effectiveness of the entire power grid infrastructure. In order to achieve this, smart meters are attached to the communication network, collecting fine-granular data. Unfortunately, as the smart meters are limited devices connected to the network and running software, they also make the whole smart grid more vulnerable than the traditional grids in term of software problems and even possible cyber attacks. In this paper, we work towards an increased software security of smart metering devices and propose a fuzzing framework, eFuzz, built on the generic fuzzing framework Peach to detect software problems. eFuzz tests smart metering devices based on the communication protocol DLMS/COSEM, the standard protocol used in Europe, for possible faults. Our experiments prove the effectiveness of using an automated fuzzing framework compared to resource demanding, human made software protocol inspections. As an example, eFuzz detected between 10 and 40 bugs in different configurations in less than 3 hours while a manual inspection takes weeks. We also investigate the quality of the eFuzz results by comparing with the traditional non-automated evaluation of the same device with respect to scope and efficiency. Our analysis shows that eFuzz is a powerful tool for security inspections for smart meters, and embedded systems in general.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Automated Testing, Fuzz Testing, Software Vulnerabilities

General Terms

Security

1. INTRODUCTION

Critical infrastructures are essential assets to the correct functioning of the economy and society. One of these infrastructures is the energy grid, which is one of the most fundamental and visible commodities as any disruption in its supply can have profound effects on virtually everything else. Currently, the energy grid is undergoing significant transformation in the direction of smart grids that will enable a new range of applications, including but not limited to fine grained tariffs to decrease consumption fluxes, remote access to metering data and better management of micro-generation.

The evolution in the energy sector has undoubtedly potential to improve the way society makes use of electricity however, it also opens the door to a new class of threats. As the current devices gain networking capabilities and the ability to execute remote commands they also become more exposed and more attractive to adversaries. Therefore, it is important to perform security audits of the different grid components.

Particularly, networked controllable electricity meters, typically referred to as smart meters, are a fundamental part of the smart grid. Smart meters are situated in each household, hence are broadly available and can be easily probed by many people. As the physical protection is limited, it is safe to assume that curious customers will inspect the devices to try and understand how they work. Furthermore, its communication capabilities would also enable an adversary of gaining control remotely. If a vulnerability is found that, for example, allows tampering with the reported usage, it can have important repercussions. The impact will be significant if the vulnerabilities are published online, thus endangering a large installation base of meters to be exploited. Since the smart meters are relatively new and there is little

standardization with respect to security, many different vendors develop their own software, and we envision that smart meter software are vulnerable to attacks.

Some of the plausible attack scenarios are as follows.

- *Disclosure of sensitive data* - Vulnerabilities in the smart meter that result in the leakage of consumption information can be used to ascertain if the home or shop owners are away.
- *Meter tampering* - Attackers may find desirable to inflate the utility bill of victims or unlawfully reduce their own.
- *Pivoting* - Penetration of meter defenses can be used for reconnaissance purposes or exploitation of other parts of the system. If the meter is connected to a domestic network the hacker could use the meter to penetrate other devices in the house. Or, more creatively, be used to make free calls or grant Internet connectivity via the GSM connection that is common in smart meters. Alternatively it can reveal valuable information about sensitive parts of the smart grid network.
- *Remote command execution* - If an adversary gains complete access to a meter, it may be used to extort individual victims, similarly to ramsonware. Meters could be switched off, or internal security mechanisms be silently disabled so the electric installation or the meter itself would be damaged.
- *Grid destabilization* - Given the remote accessibility of smart meters, an attacker could gain controller of hundreds or thousands of identical metering devices and exploit vulnerabilities to simultaneously switch heavy loads such as electric cars on and off, with the intent to destabilize and bring down the power grid.

One technique that is widely used to find vulnerabilities in proprietary software is fuzzing. Very succinctly, it consists of trying numerous combinations of inputs, violating protocol or file format rules, to see how the target responds to unexpected requests. The simplest way to perform a fuzzing attack is manually. First one develops a crude version of a protocol client and then chooses a set of values the tester believes may cause software faults in the target system. However, this approach has various limitations: it is hard to fuzz complex protocols, reuse of code is not always possible, and the developers have to focus on more than the target protocol. To cope with such limitations, fuzzing frameworks have been developed, such as SPIKE [1], SNOOZE [2] and Peach [7].

These fuzzing frameworks typically target software applications running on traditional computing platforms, such as desktops. To monitor the working state of the targeted software, they might for example attach debuggers to the process or inspect memory consumption. Unfortunately, in embedded systems like smart meters, these capabilities are not always available and the fuzzing framework is usually physically separated from the investigated target. Therefore, to fuzz an embedded system like a smart meter, it is necessary to use a framework that supports the communication protocols available in the target and that allows specific types of monitoring developed for limited devices.

In this paper, we address this void and present a new fuzzing framework, developed particularly for smart meters. The main purpose of eFuzz is to investigate vulnerabilities in smart meter communication protocols. In the European market, DLMS/COSEM is the *de facto* communication protocol for smart metering application, hence we present this protocol as a use case in this paper to conduct our tests.

We provide an abstract, customizable and versatile fuzzing framework for smart meters that is not restricted to any specific device. When evaluating the functionality and effectiveness of our framework, we present the results of a vulnerability test performed on a smart meter currently deployed in the Netherlands. The tests conducted on this specific smart meter are also compared with traditional approach, analysis made by a human inspector, and the results are given.

The outline of this paper is as follows: Section 2 highlights relevant related work. Section 3 introduces fundamentals of fuzzing, the DLMS/COSEM communication protocol and our proposed prototype eFuzz. Section 4 evaluates the performance of our automated fuzzing approach and discusses the effectiveness of different fuzzing strategies. Finally, we conclude in Section 5.

2. RELATED WORK

Fuzzing was originally developed at the University of Wisconsin in 1988 [15] and as one of the first applications, command-line UNIX programs were tested [16]. The success of this technique was astonishing as this approach induced crashes in 25 to 33% of test cases. Since then, fuzzing has grown in sophistication and scope. Recent literature describes several techniques such as taint analysis that aim to increase the efficiency of fuzzers [3, 4, 20].

In essence, this technique first marks untrusted data (*e.g.* input forms) as tainted. Then it tracks its propagation and which variables are affected by it during runtime. This mark and track method exposes the flows between data sources and sinks. Such data progress information can be used to identify the most relevant paths of execution, which can later be used to improve the efficiency of the fuzzer. Another important component is being able to monitor faults caused by the fuzzer. However, such techniques, in the current state, can not be readily applied in embedded systems. For example, in these systems it is not always possible to have access to the binaries running on the target.

With the prominence of the Internet of Things, there is a growing need to evaluate the security of embedded devices. These systems assume numerous shapes and sizes and support a myriad of protocols and interfaces. Particularly for fuzzing, the heterogeneity of targets significantly increases the cost of tests as fuzzers needs to be rewritten to support the specific protocols and interfaces available on the device under test. Currently the most common way to deal with such difficulties is to use dedicated hardware to enable compatibility, both on the interface and protocol. [5, 17] feature various examples of fuzzing setups.

Currently, available literature [11, 13, 18, 9, 8, 21] on smart grid security is focused on higher level designs or guidelines that aim to raise awareness and assist stakeholders implementing this complex network in a relatively secure way since this is a nascent field.

In comparison, few researchers have contributed tools to evaluate devices already in the market. In particular for smart meters, one of the exceptions that is freely available is Termineter [12]. Termineter is a python framework for security testing that supports ANSI C12.18 and C12.19, standards that are commonly used for smart metering in the United States. On the commercial side ProtoPredator for Smart Meters (PP4SM) [14] enables fuzzing of ANSI C12.18 meters through the optical interface.

For DLMS/COSEM, the authors are not aware of any similar applications, commercial or non-commercial, further reinforcing the need for a tool such as eFuzz.

3. eFuzz: A FUZZER FRAMEWORK FOR SMART METERS

This section first provides a high level overview of fuzzing frameworks. After a description the DLMS/COSEM protocol used in European smart meters, we present details on eFuzz and its fuzzing approach.

3.1 Fuzzing

Fuzz testing is a security testing technique used to detect vulnerabilities in software applications or network devices. It is an automated process that feeds the target with malformed data to induce malfunctions. To detect these erroneous actions, fuzzing includes monitored activities like memory consumption inspections, thrown exceptions, crashes or other unexpected behaviors.

Fuzzing is one of the premier techniques to uncover vulnerabilities through testing [3, 4] as it does not require access to the source code of the target and it has a low cost. The most common targets include network protocols and file formats.

In general, fuzzers can be divided in three categories with respect to how they generate new packets or files. The first one is to feed completely random data as input to the target. This method is simple but is less likely to yield good results. In the second category, the mutation-based fuzzers alter pregenerated valid inputs in hope of discovering a vulnerability. In the third category, generation-based fuzzers create files or packets from scratch with the same intent based on a user specified model.

Compared to others, the main advantage of the second category is that little knowledge of the file or protocol format is required and thus the preparation time is modest. However the code coverage is dependent on the coverage provided by the original valid inputs. On the other hand, constructing new files or packets can only be done if the tester has sufficient knowledge of the file or protocol format. But this approach results in a more comprehensive test case suite and thus, it is more likely to find vulnerabilities. eFuzz belongs in this category. Table 1 provides a succinct summary of these advantages and disadvantages.

Type	Advantages	Disadvantages
Random	Simple. Quick. Low cost.	Attacks only the surface. Useless with checksums. Poor coverage.
Mutation	Relatively Simple. Reusable for different SW.	Needs numerous valid inputs to get a good coverage.
Generation	Efficient (if well modeled).	Time-consuming. Requires knowledge of the file/protocol format. Reusable only with the same format.

Table 1: Advantages and disadvantages of the different fuzzing approaches. Adapted from [4].

There are only a few fuzzing frameworks. Dave Aitel proposed a block-based fuzzing framework designated SPIKE [1]. The framework was developed to simulate network protocol clients and automate black-box testing. It is implemented as a C-like API and scripting language. These are used to leveraged the tester's knowledge of the protocol.

SNOOZE [2] is a network protocol fuzzer that implements a stateful fuzzing approach. Therefore, a tester can describe the stateful operation of the protocol and specify the messages to be generated in each state. Moreover, it provides fuzzing primitives that are specific to certain attacks and thus allow the tester to direct her efforts solely on a class of vulnerabilities.

Peach [7], the brainchild of Michael Eddington, has become one of the most mature and, arguably, the most widely used fuzzing framework. Hence this framework was chosen for the development of eFuzz. Peach is capable of both generation and mutation based fuzzing. To define the structure, type information and relationships in the data to be fuzzed, Peach uses manually crafted Peach Pit files. These files describe the protocol model and define how such model will be used to test the target.

The modular architecture of Peach encourages code reuse and provides easy extendability. The main advantages of using Peach instead of developing a fuzzer from the ground up are the abstraction of the mutation strategies, the simple way the protocol can be modelled and the modularity of the various components (custom or built-in).

3.2 DLMS/COSEM Communication Protocol

The Device Language Message Specification (DLMS) consists of a general concept for abstract modelling of communication entities. On top of this, the *COmpanion Specification for Energy Metering* (COSEM) provides a set of standards that define the rules for data exchange between smart grid devices, such as an energy meter and a data accumulator. Together they provide several features, such as (a) an object model to view and access the different functionalities of a

meter, (b) an identification system for all data, (c) a method for communicating with the model and (d) a transport layer to accommodate the information flows between the meter and other devices. The protocol is based on the *Open System Interconnection* (OSI) seven-layer model. However in the case of smart meters they are practically collapsed in four - physical, data link, transport and application layers.

In a similar fashion to the OSI model, the physical layer in DLMS/COSEM defines how to transfer information to and from the meter. The data layer provides the messaging methods to modify data and communicate with the device. The transport layer enables data transfer based on the IPv4 network. Finally the application layer represents the functional aspects of the energy meter so applications can access them.

Prior to exchanging metering information an association must be set up, initiated by the client through the object model interface. From that moment, the server is also able to send notification without an explicit request. DLMS/COSEM supports authentication and confidentiality services based on symmetric key encryption.

As explained above DLMS/COSEM is a connection oriented protocol and it encapsulates its traffic differently depending on the interface. Since eFuzz uses a physical connection the following explanations will focus on the layers for this particular configuration. Figure 1 features an high-level overview of DLMS/COSEM request over serial port.

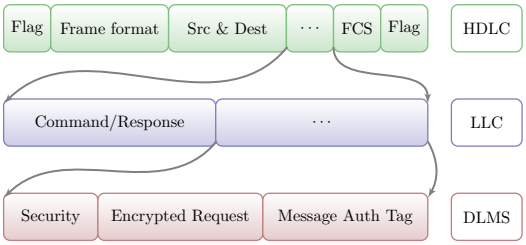
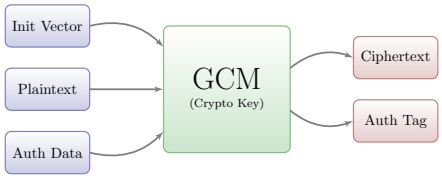


Figure 1: DLMS/COSEM packet for direct connections.

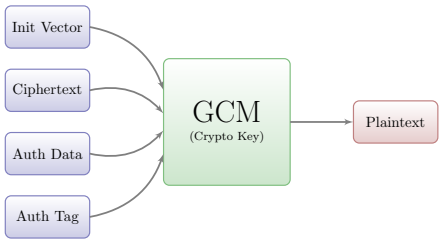
The outer layer is the physical layer, not shown in the figure. For the current set-up it consists of the serial communication over USB. When using a direct connection in DLMS/COSEM, the High-Level Data Link Control (HDLC) protocol is used. This data link layer, show in green, is composed of opening and closing flags, the type of frame, source and destination identifiers, payload and checksum (Frame Check Sequence or FCS) to ensure data integrity. The Logical link control (LLC), colored blue, is the upper sub-layer of the data link layer. It is short in length and is used to specify is the payload is a response or a command. The application layer, tinted red, contains the DLMS protocol data unit. Its content specifies the level of security it is using and, if applicable, proceeds to transmit the encrypted request followed by the authentication tag.

DLMS/COSEM supports three access security levels to allow the client to read data. First, no authentication is necessary to access the device. Second, in the low level (LLS) mode, the client authenticates using a password to the device. Third, in the high level security mode (HLS) access is validated in both directions. This mode can use four different algorithms, MD5, SHA-1, GMAC (Galois Message Authentication Code) or a secret method known only by the meter and the client.

In addition to client authentication, the data transport can be encrypted. All possible four combinations of encryption and authentication are supported. In case messages are both encrypted and authenticated, only one algorithm is allowed: Galois Counter Mode (GCM) with AES-128. Figure 2 depicts how encrypted and authenticated packets are generated.



(a) Encryption



(b) Decryption

Figure 2: Encryption and authentication of DLMS Application Layer Protocol Data Units (APDUs).

The initialization vector is the concatenation of the system title and frame counter. The former is device specific and is announced in the configuration phases of the session. While the latter is transmitted alongside the ciphertext. The key for the Galois Counter Mode is specific for each device, in other words, both the meter and the PC have their own. Lastly, the authentication data includes the symmetric authentication key. The decryption process is similar but adds a verification steps for the authentication tag.

It is also insightful to have an understanding of the complete communication flow that a simple request entails. A generic example over physical connection is show in Figure 3. Serial connection is divided in three distinct phases. First, a handshake takes place where the meter identifies itself and announces the serial configuration it supports, including baudrate and other serial related parameters. Second, the client (PC) initializes the HDLC (High-Level Data Link Control, a data link layer protocol) link by transmitting a frame to the meter. After reception the meter replies with the frame configurations it supports. The third and last step to complete the connection establishment is the pair AARQ/AARE (Application Association Request and Response, respectively). The former is sent by the PC and the latter is the response

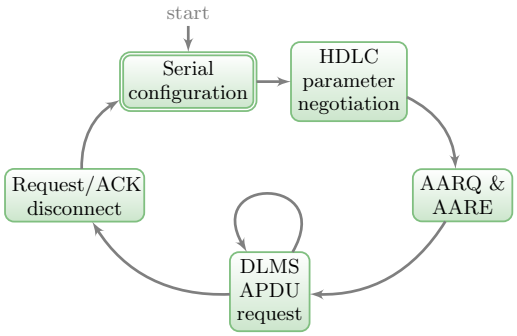


Figure 3: A complete DLMS/COSEM flow including establishing a connection, making a request, reading the response and closing the connection. This example is for the physical interface.

of the meter. This exchange is used to establish more parameters specific to the DLMS layer (*i.e.* independent of the outer-layers which depend on the interface). If defined in the security configurations, these are the first packets to be encrypted and authenticated.

Finally, we are able to start the actual DLMS/COSEM communication. In this session, the client may query the server for usage data, time and date, or modify the parameters of the meter, such as increase the number of available tariffs, clear the logs, etc. Although the previous steps may change from meter to meter and from interface to interface, the DLMS *Application Layer Protocol Data Units* (APDUs) do not. This makes this layer the most interesting point of attack for tools like eFuzz as it is prevalent across compliant meters. Therefore, this makes the application very versatile as it is simple to adapt it to different devices.

3.3 eFuzz

As previously mentioned eFuzz is built on top of Peach (version 2.3.9) to facilitate the development of the fuzzer.

Peach is divided in different components according to their task which are all orchestrated by the aforementioned Pit file. In this context, the most relevant ones are the "transformers", "fixups" and "publishers". Peach comes pre-installed with several instances of each component. However, since embedded systems are not typical targets for fuzzing, it was necessary to develop custom versions to communicate with the meter under test.

According to [6], transformers perform static modifications or encoding of the parent elements. In eFuzz, a custom transformer was written to encrypt the APDUs. Fixups are similar to transformers but instead operate on data retrieved from an another element other than their parent. When generating DLMS/COSEM packets, a fixup is used to determine the frame check sequence (FCS). A reference implementation in C can be found in [19]. Finally, publishers are responsible for the I/O communication, in this case serial. eFuzz uses the PySerial [10] library to enable cross-platform serial communication. Figure 4 illustrates the aforementioned components as well as their interactions and dependencies.

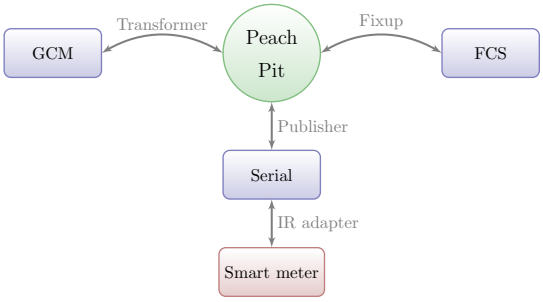


Figure 4: Different components used in eFuzz/Peach and their relationship.

As explained, the transformer is used to encrypt the mutated plaintext to the final form. In the current context, all necessary keys and passwords to encrypt and decrypt the text were available. Presuming access to this data can be acceptable or not depending on the context. For an "ordinary" attacker, it is not a realistic premise. Moreover, since every meter should be deployed with its own keys, in case of key disclosure only one device would be compromised. However, in the context of a security evaluation this is an acceptable assumption. For example, a company hired to assess the security of the device would be given access to the keys and perhaps the firmware source code. Then, they would use eFuzz in an initial approach to autonomously detect possible points of interest. The respective report would serve as a guide to security experts and complement a full featured source code review. Moreover smart meter manufactures can also greatly benefit from eFuzz by integrating it in their test suites, thus detecting flaws early in the product development process.

Generating packets

eFuzz uses a modular approach to generate the fuzzed requests. The most abstract data model is a common wrapper. The block incorporates the HLDC and the LLC layers. Some of the fields are randomly chosen while others, such as size and checksums, are computed deterministically. The checksum, also known as frame check sequence, is handled by a custom fixup instance. The DLMS payload is declared as well, but the actual contents are not specified.

There are two distinct models describing the APDUs sent from the PC to the meter.

1. The first APDU "inherits" the common wrapper and only replaces the necessary fields. The request is fuzzed at the plaintext level and consequently encrypted, through a user-defined transformer. This APDU is modeled after a request that queries the meter for the local time and date.

2. The second APDU defined by eFuzz is only used as a reference. It is similar to the previous one but none of the fields are modified.

The two APDUs that correspond to the answers from the meter serve different purposes.

1. The response to the fuzzed request is read but disregarded. The sole purpose of the first APDU is to provoke faults in the target therefore the response has little relevance.
2. The following response is used to assess the health of the meter. Since it corresponds to a reference request can be compared with a reference response. If the received and expected data do not match, eFuzz logs the event.

There is a myriad of strategies built-in Peach that define how the model is mutated. The existing mutation strategies are divided in two main categories: sequential and random.

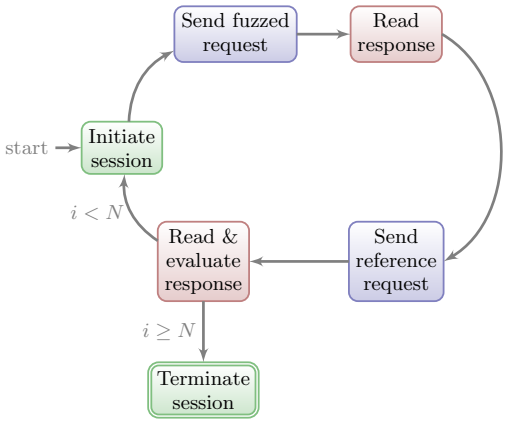
As indicated by the name, sequential mutations are deterministic and progress from the first to the last fuzzable field. Each field is fuzzed only once. Within this category there is a distinction between linear (`SequentialMutationStrategy`), and random progress (`Random DeterministicMutationStrategy`) . These approaches are similar with only the order changing, which may allow the fuzzer to detect flaws earlier on the test suite.

On the other hand, random strategies are not finite and fuzz up to N elements per iteration which allows for more flexibility and bigger coverage since combinations of fields are also fuzzed. The downside is the running time. Therefore, a typical test starts with sequential strategies and then proceeds to random ones. For reproducibility purposes, it is also possible to specify a seed for the internal random number generator.

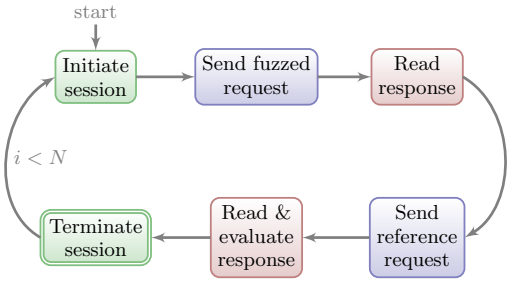
In this category there are three mutators: `RandomMutationStrategy`, `SingleRandomMutationStrategy` and `DoubleRandomMutationStrategy`. The only distinction between the first and the others is the maximum number of elements that are fuzzed per iteration, which is 7 instead of 1 or 2. Nonetheless for `RandomMutationStrategy` the maximum value can be arbitrarily chosen. Furthermore, it is possible for developers to program their own strategies.

The response of the meter to the invalid queries is not necessarily relevant as there is not correct or incorrect response. More interesting is seeing how they might affect the meter behavior thereafter. Given the specificities of DLMS/COSEM two main ways were chosen to do so (see Figure 5).

The first approach is to send a reference request after each fuzzed packet. If the response matches what is expected, the meter is functioning properly. Otherwise, this behavior should be signalled and stored for human inspection. Thereafter the process is repeated. An exception to this cycle occurs when the target does not respond as expected. In this situation the session is restarted. Otherwise, the subsequent tests would not be evaluated by the meter as it was already in a faulty situation. As an example, assume request i causes the meter to reach an unknown state that rejects all further communication in the session. Then even if request $i + 1$ would normally trigger a new problem it would go unnoticed. The state machine describing this behavior is show in Figure 5a.



(a) Fuzzed and reference packets are all sent within the same session.



(b) The session is restarted after each fuzzing attempt.

Figure 5: State machines describing the two approaches used to fuzz the target device. On these diagrams i indicates the current fuzzing iteration while N is the total number of tests available. In both situations the first iteration uses a reference request to ensure the target is in a known state.

The second approach is similar but the session is restarted every time a fuzzing attempt terminates, as portrayed in Figure 5b.

These contrasting approaches may yield different results. The former is expected to detect more memory leaks or other memory related problems, for example, a memory leak triggered by an attack may not be mediately noticeable and only have consequences later on. If the session was terminated in between, there would be a chance for the meter to clear its state and recover from such a problem.

4. RESULTS

4.1 Detection Performance of eFuzz

To assess the performance of eFuzz, this Section presents the amount of unexpected behaviors provoked in the target. As explained before, after sending each fuzzed request, eFuzz sends a reference query and compares the result with the

correct response. If these values do not match, the dialog is stored as an unexpected behavior and the session is restarted before continuing.

Due to the detail used to model the protocol, the complete test suite of eFuzz when using the sequential strategies includes 54727 distinct cases. Running all tests requires around 7 days. However to be able to obtain data for different configurations, within a reasonable time frame, a subset of 500 tests was used. The subset completes within approximately 3 hours.

Table 2 shows the number of unexpected behaviors induced with three distinct strategies. Sequential Random refers to the aforementioned `RandomDeterministicMutationStrategy` class. The other two use random strategies where n specifies the maximum number of fields that are fuzzed per iteration.

Mutation strategy	Number of unexpected responses
Sequential Random	40
Random $n = 2$	21
Random $n = 7$	10

Table 2: Overview of the number of unexpected responses caused by eFuzz discriminated by strategy. Variable n indicates the maximum number of fields that are mutated in each iteration. In the sequential strategy this value is by definition one.

The amount of invalid responses is significant, in particular considering the short amount of time it required. The sequential strategy was the most proliferous, reinforcing the point made above that this strategy should be used first.

In addition to the quantity, it is also insightful to understand what the meter response actually was. This information is condensed in Table 3. The responses are separated by exception-response, frame reject (specific DLMS/COSEM messages) or other.

The Exception-Response APDU is an optional APDU sent by the COSEM server application layer that indicates to the client application layer the service requested could not be processed. Particularly for the case shown, the request was not processed as access to the service was not allowed. Alternatively the server can simply discard the message without sending this APDU.

The Frame Reject (FRMR) is used to report one of various conditions occurred and retransmission of the identical

Mutation Strategy	Exception-response	Frame Reject	Other
Sequential Random	50%	50%	0%
Random $n = 2$	57%	33%	10%
Random $n = 7$	50%	40%	10%

Table 3: Distribution of the unexpected meter responses by cause.

Parameter	eFuzz	Manual
Time	Hours/few days	Weeks
Coverage	DLMS/COSEM logical attacks	Physical attacks. Weaknesses in configurations, key storage, retrieval.
Quantity findings	High	Low
Quality findings	Low	High

Table 4: Comparison of automated and non-automated approaches to detecting smart meter vulnerabilities.

frame can not correct it. The situations that may trigger an FRMR include the receipt of an undefined or not implemented command or response, receipt of a frame that is too long, receipt of an invalid frame, or receipt of a frame containing a field that is not allowed.

Categorizing the meter responses shows a clear majority of `exception-response` and `frame-reject` APDUs. These messages are used to signal errors and should not occur in response to valid requests, which suggests one of the previous messages perturbed the internal state of the server and has caused undesirable side effects.

4.2 Discussion

The results presented in Section 4.1 indicate this preliminary work exposes bugs that would probably go unnoticed otherwise. Just like in traditional software application, fuzzing is a good technique to test proprietary code. The approach used can generate combinations of inputs that would not be possible by a human tester as she would inclined to follow the same assumptions as the original developers.

A series of experiments performed on the target were able to provoke between 10 and 40 incorrect responses, depending on the mutation strategy selected, in a short period of time. Although with this approach it is not possible to automatically determine if these bugs have security consequences, it pinpoints areas of interest that a human expert can focus on.

Nevertheless, this method is not meant to replace professional human auditors but instead complement them. The added value is accomplished by means of an automated tool that exhaustively finds all possible bugs with respect to diverse inputs. Note that manual inspection by a human expert cannot achieve the same level of testing within an acceptable time frame.

To better understand the complementarity of both approaches, the results of eFuzz and a security evaluation made manually are compared. Table 4 compares the two with respect to four different parameters.

One of the main advantages of fuzzing is its automated na-

ture. Automation makes it faster, hours or days instead of weeks, than manual evaluations and requires little human supervision. On the other hand eFuzz has a very specific coverage while a manual approach can be much broader and detect classes of vulnerabilities that a fuzzer can not, for example if keys are stored securely or if permissions are properly enforced. As previously shown eFuzz is able to detect a large quantity of findings while manual inspections are typically more selective. This disparity is reflected in the quality of each discovery. Issues uncovered by fuzzing need to be inspected by an expert to determine if they constitute a real security threat.

5. CONCLUSION

This paper describes eFuzz, a DLMS/COSEM fuzzer built on top of Peach to complement security evaluations of smart meters. It works autonomously by generating invalid or malformed protocol packets derived from a model of the standard. By constantly sending these requests to the meter followed by a reference query, one can assess if the fuzzed data was able to interfere with the normal behavior of the target.

In the preliminary tests with a electricity meter currently in the market using a direct optical interface, eFuzz was able to perturb the correct functioning of the target. In less than 3 hours, the tests revealed between 10 and 40 issues depending on the mutation strategy. The majority of the invalid responses indicated that the client was trying to access a service that was not allowed or that the request was malformed, which suggests previous messages caused the meter to reach an invalid state.

The obtained results show that fuzzing approach is an efficient way to detect flaws in the protocol implementation given the limited human effort it requires. Additionally, our framework for fuzzing embedded systems, eFuzz, will provide researchers the capability to apply fuzzing techniques as a vital step for the evaluation of smart meters and other networked embedded systems. Ultimately, we hope that eFuzz will make smart meter communications harder to penetrate, and consequently elevate the bar for the security of critical infrastructures.

As alluded to previously, there is ample room for improvements in eFuzz. For example, in its current version only a small part of the DLMS/COSEM is modelled. To achieve better results, it would be beneficial to increase the coverage of the standard and consequently of the firmware code. Furthermore considering more parameters, such as the latency of the responses, when assessing the health of the meter is also a possible improvement. Another advancement that is currently being worked on is exploring other communication interfaces, notably GSM, since not all meters are equipped with an optical port. Finally, at this time, the main fuzzers readily available are not optimized for embedded systems. As networking capabilities are introduced to a growing number of micro-controller based devices, it is fundamental to ensure security is preserved. Fuzzing can play a crucial role in this task, but first it needs to expand its capabilities to these new classes of targets.

6. REFERENCES

[1] D. Aitel. The advantages of block-based protocol analysis for security testing. *Immunity Inc., February*, 2002.

[2] G. Banks, M. Cova, V. Felmetsger, K. Almeroth, R. Kemmerer, and G. Vigna. SNOOZE: toward a Stateful NetwOrk prOtocol fuzZEr. In *Information Security*, pages 343–358. Springer, 2006.

[3] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier. Finding Software Vulnerabilities by Smart Fuzzing. *The Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 427–430, Mar. 2011.

[4] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier. A taint based approach for smart fuzzing. *The Fifth IEEE International Conference on Software Testing, Verification and Validation*, pages 818–825, Apr 2012.

[5] Deja vu Security. Fuzzing Embedded Devices with Peach Fuzzer. `https://www.youtube.com/watch?v=yevXIDaI_SA`. Accessed: 2014-06.

[6] Deja vu Security. Peach fuzzing platform. `http://old.peachfuzzer.com/v2/peach23.html`. Accessed: 2014-02.

[7] Deja vu Security. What is Peach? `http://old.peachfuzzer.com/WhatIsPeach.html`. Accessed: 2014-03.

[8] Z. Fan, P. Kulkarni, S. Gormus, C. Efthymiou, G. Kalogridis, M. Sooriyabandara, Z. Zhu, S. Lambotharan, and W. H. Chin. Smart grid communications: Overview of research challenges, solutions, and standardization activities. *Communications Surveys & Tutorials, IEEE*, 15(1):21–38, 2013.

[9] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke. A survey on smart grid potential applications and communication requirements. *Industrial Informatics, IEEE Transactions on*, 9(1):28–42, 2013.

[10] C. Liechti. pySerial's documentation. `http://pyserial.sourceforge.net`. Accessed: 2014-06.

[11] P. McDaniel and S. McLaughlin. Security and privacy challenges in the smart grid. *IEEE Security and Privacy*, 7(3):75–77, May 2009.

[12] S. J. McIntyre. Termineter. `https://github.com/securestate/termineter`. Accessed: 2014-06.

[13] A. R. Metke and R. L. Ekl. Security technology for smart grid networks. *Smart Grid, IEEE Transactions on*, 1(1):99–107, 2010.

[14] MicroSolved Inc. Protopredator. `http://microsolved.com/protoPredator.html`. Accessed: 2014-06.

[15] B. Miller. CS 736 Fall 1988 Project List, 1988.

[16] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, Dec. 1990.

[17] MWR InfoSecurity. Usb fuzzing for the masses. `https://labs.mwrinfosecurity.com/blog/2011/07/14/usb-fuzzing-for-the-masses/`. Accessed: 2014-02.

[18] A. J. Paverd and A. P. Martin. Hardware security for device authentication in the smart grid. In *Smart Grid Security*, pages 72–84. Springer, 2013.

[19] W. Simpson. PPP in HDLC-like framing. July 1994. RFC 1662.

[20] T. Wang, T. Wei, G. Gu, and W. Zou. TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. *IEEE Symposium on Security and Privacy*, pages 497–512, 2010.

[21] W. Wang and Z. Lu. Cyber security in the smart grid: Survey and challenges. *Computer Networks*, 57(5):1344–1371, 2013.

BIBLIOGRAPHY

[1] S. Feuerhahn, M. Zillgith, C. Wittwer, and C. Wietfeld, *Comparison of the communication protocols DLMS/COSEM, SML and IEC 61850 for smart metering applications,* 2011 IEEE International Conference on Smart Grid Communications (SmartGridComm) , 410 (2011).

[2] G. Banks, M. Cova, and V. Felmetsger, *SNOOZE: toward a Stateful NetwOrk prOtocol fuzZEr,* Information ... , 343 (2006).

[3] D. McGrew and J. Viega, *The galois/counter mode of operation (gcm),* Submission to NIST. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf (2004).

[4] EEI-AEIC-UTC, *Smart Meters and Smart Meter Systems: A Metering Industry Perspective,* (2011).

[5] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier, *A Taint Based Approach for Smart Fuzzing,* 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation , 818 (2012).

[6] R. Anderson and S. Fuloria, *Smart meter security: a survey,* cl.cam.ac.uk , 1.

[7] Parliament, The European and Union, The Council Of The European, *Directive 2009/72/EC of the European Parliament and of the Council of 13 July 2009 concerning common rules for the internal market in electricity and repealing Directive 2003/54/EC Text with EEA relevance,* (2009).

[8] M. Jawurek and F. Freiling, *Privacy threat analysis of smart metering,* ... of the 41th Annual Conference of the ... (2011).

[9] D. Aitel, *The advantages of block-based protocol analysis for security testing,* Immunity Inc., February (2002).

[10] Dejavu Security, *What is Peach?* (2014).

[11] P. McDaniel and S. McLaughlin, *Security and privacy challenges in the smart grid,* IEEE Security and Privacy **7**, 75 (2009).

[12] A. R. Metke and R. L. Ekl, *Security technology for smart grid networks,* Smart Grid, IEEE Transactions on **1**, 99 (2010).

[13] A. J. Paverd and A. P. Martin, *Hardware security for device authentication in the smart grid,* in *Smart Grid Security* (Springer, 2013) pp. 72–84.

[14] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, *A survey on smart grid potential applications and communication requirements,* Industrial Informatics, IEEE Transactions on **9**, 28 (2013).

[15] Z. Fan, P. Kulkarni, S. Gormus, C. Efthymiou, G. Kalogridis, M. Sooriyabandara, Z. Zhu, S. Lambotharan, and W. H. Chin, *Smart grid communications: Overview of research challenges, solutions, and standardization activities,* Communications Surveys & Tutorials, IEEE **15**, 21 (2013).

[16] W. Wang and Z. Lu, *Cyber security in the smart grid: Survey and challenges,* Computer Networks **57**, 1344 (2013).

[17] M. HADLEY, N. Lu, and A. DEBORAH, *Smart-grid security issues,* IEEE Security and Privacy **8**, 81 (2010).

[18] C4Security, *The Dark Side of the Smart Grid - Smart Meters (in) Security*, Tech. Rep.

[19] M. Gröne and M. Winandy, *Applying a Security Kernel Framework to Smart Meter Gateways,* ISSE 2012 Securing Electronic Business Processes , 252 (2012).

[20] J. Chinnow, K. Bsufka, A.-D. Schmidt, R. Bye, A. Camtepe, and S. Albayrak, *A simulation framework for smart meter security evaluation,* 2011 IEEE International Conference on Smart Measurements of Future Grids (SMFG) Proceedings , 1 (2011).

[21] A. Lunkeit, T. Voß, and S. Augustin, *Threat Modeling Smart Metering Gateways,* **9**, 5 (2013).

[22] C. Rottondi, G. Verticale, A. Capone, P. Milano, and P. Leonardo, *A security framework for smart metering with multiple data consumers,* ... INFOCOM WKSHPS), 2012 ... (2012).

[23] O. Tan, D. Gunduz, and H. Poor, *Increasing smart meter privacy through energy harvesting and storage devices,* Selected Areas in ... (2013), arXiv:arXiv:1305.0735v1 .

[24] R. Anderson and S. Fuloria, *Who controls the off switch?* Smart Grid Communications (... (2010).

[25] DLMS User Association, *Official DLMS website,* (2014).

[26] G. Struklec and J. Marsic, *Implementing DLMS/COSEM in smart meters,* ...2011 8th International Conference on the ... , 747 (2011).

[27] N. Calamaro, E. Abramowitz, and Y. Beck, *A General Review on DLMS / COSEM and IEC 60870-5 Smart Grid Standards,* , 1 (2012).

[28] Gurux, *Gurux Website,* (2014).

[29] OpenMUC, *jDLMS Documentation website,* (2014).

[30] Kalkitech, *SYNC 500 series - DLMS COSEM Protocol Library,* (2014).

[31] T. Instruments, *Application Report: Using TI's DLMS COSEM Library,* (2013).

[32] B. Liu, L. Shi, Z. Cai, and M. Li, *Software Vulnerability Discovery Techniques: A Survey,* 2012 Fourth International Conference on Multimedia Information Networking and Security , 152 (2012).

[33] S. McLaughlin, D. Podkuiko, S. Miadzvezhanka, A. Delozier, and P. McDaniel, *Multi-vendor penetration testing in the advanced metering infrastructure,* Proceedings of the 26th Annual Computer Security Applications Conference on - ACSAC '10 , 107 (2010).

[34] B. Miller, *CS 736 Fall 1988 Project List,* (1988).

[35] B. P. Miller, L. Fredriksen, and B. So, *An empirical study of the reliability of UNIX utilities,* Communications of the ACM **33**, 32 (1990).

[36] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier, *Finding Software Vulnerabilities by Smart Fuzzing,* 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation , 427 (2011).

[37] T. Wang, T. Wei, G. Gu, and W. Zou, *TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection,* 2010 IEEE Symposium on Security and Privacy , 497 (2010).

[38] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, *Prospex: Protocol Specification Extraction,* 2009 30th IEEE Symposium on Security and Privacy , 110 (2009).

[39] OpenRCE: The Open Reverse Code Engineering, *Sulley: A pure-python fully automated and unattended fuzzing framework.* (2013).

[40] P. Amini and A. Portnoy, *Fuzzing sucks!* (2007).

[41] Oulu University Secure Programming, *PROTOS - Security Testing of Protocol Implementations,* (2003).

[42] Oulu University Secure Programming, *PROTOS Protocol Genome Project,* (2007).

[43] R. Leevi and S. Puupera, *Domain Model Based Black Box Fuzzing Using Regular Languages*, Ph.D. thesis, University of Oulu (2010).

[44] A. Helin, J. Viide, M. Laakso, and J. Röning, *Model Inference Guided Random Testing of Programs with Complex Input Domains,* (2006).

[45] J. Viide, A. Helin, M. Laakso, and P. Pietikäinen, *Experiences with Model Inference Assisted Fuzzing.* WOOT (2008).

[46] P. Godefroid, *Random Testing for Security : Blackbox vs . Whitebox Fuzzing,* ... of the 2nd international workshop on Random testing: ... , 59593 (2007).

[47] P. Godefroid, M. Levin, and D. Molnar, *Automated Whitebox Fuzz Testing.* NDSS (2008).

[48] P. Godefroid, A. Kiezun, and M. Levin, *Grammar-based whitebox fuzzing,* ACM Sigplan Notices , 206 (2008).

[49] P. Godefroid, M. Y. Levin, and D. Molnar, *SAGE: Whitebox Fuzzing for Security Testing,* Queue **10**, 20 (2012).

[50] P. Godefroid, *Automated Whitebox Fuzz Testing with SAGE,* (2009).

[51] P. Amini and A. Portnoy, *Sulley : Fuzzing Framework*, Tech. Rep.

[52] R. McNally, K. Yiu, D. Grove, and D. Gerhardy, *Fuzzing: The State of the Art,* (2012).

[53] Y. Lu, W. Lifa, P. Fan, Z. Honglin, and H. Zheng, *Automatic Fault Localization for Fuzzing,* 2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control , 388 (2011).

[54] M. Vuagnoux, *Autodafe: An act of software torture,* 22nd Chaos Communications Congress, Berlin, . . . , 1 (2005).

[55] DLMS User Association, *COSEM Interface Classes and the OBIS Identification System, "Blue Book"* (2010), ed. 10.0.

[56] DLMS User Association, *DLMS/COSEM Architecture and Protocols, "Green Book"* (2009), ed. 7.0.

[57] DLMS User Association, *DLMS/COSEM Conformance Testing Process, "Yellow Book"* (2010), ed. 3.0.

[58] DLMS User Association, *COSEM Glossary of Terms, "White Book"* (2003), ed. 1.0.

[59] M. Sutton, A. Greene, and P. Amini, *Fuzzing: brute force vulnerability discovery* (Pearson Education, 2007).

[60] Deja vu Security, *Peach fuzzing platform,* http://old.peachfuzzer.com/v2/peach23.html, accessed: 2014-02.

[61] S. J. McIntyre, *Termineter,* https://github.com/securestate/termineter, accessed: 2014-06.

[62] MicroSolved Inc., *Protopredator,* http://microsolved.com/protoPredator.html, accessed: 2014-06.

[63] R. Housley, *Using aes-ccm and aes-gcm authenticated encryption in the cryptographic message syntax (cms),* (2007).

[64] M. Dworkin, *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC* (US Department of Commerce, National Institute of Standards and Technology, 2007).

[65] W. Simpson, *PPP in HDLC-like framing,* (1994), rFC 1662.

[66] C. Liechti, *pySerial's documentation,* http://pyserial.sourceforge.net, accessed: 2014-06.

[67] H. Rodriguez, *From fuzzing to 0-day,* http://blog.techorganic.com/2014/05/14/from-fuzzing-to-0-day/, accessed: 2014-06.

[68] ABB, *RER620 IEC 60870-5-101/104 Communication Protocol Manual*, Tech. Rep. (2011).

[69] D. Vyas and H. Pandya, *Advance Metering Infrastructure and DLMS/COSEM Standards for Smart Grid,* International Journal of Engineering Research & ... **1**, 1 (2012).

[70] I. software, *DLMS communication packages provided by ,* (2014).

[71] A. P. Vidal, *upcommons.upc.edu*, Ph.D. thesis, Universitat Politécnica de Catalunya (2012).

[72] J. Butts and S. Shenoi, *IFIP Advances in Information and Communication Technology* (2011).

[73] M. Eddington, *Black Hat Europe*, Tech. Rep. (2009).

[74] C. Miller, *Babysitting an Army of Monkeys: An analysis of fuzzing 4 products with 5 lines of Python,* (2010).

[75] P. Oehlert, *Violating assumptions with fuzzing,* Security & Privacy, IEEE (2005).

[76] I. V. Sprundel, *Fuzzing: Breaking software in an automated fashion,* Talk at: 22nd Chaos Communication Congress: ... , 1 (2005).

[77] D. Aitel, *An introduction to SPIKE, the fuzzer creation kit,* presentation slides), Aug (2002).

[78] T. Nicol, *Privacy protection through limited load signal distortion*, Ph.D. thesis (2011).

[79] G. Tyler, *Look out ! It's the fuzz,* IA newsletter - The Newsletter for Information Assurance Technology Professionals **10**, 1 (2007).

[80] T. Morris, R. Vaughn, and E. Sitnikova, *Advances in the protection of critical infrastructure by improvement in industrial control system security,* ... Security Conference-Volume 138 , 67 (2013).

[81] P. Amini, *Fuzzing Frameworks,* (2007).

[82] Q. Lanlan, X. Dan, W. Zhiyong, and X. Qixue, *New Development of Fuzzing-based Vulnerabilities Mining Research,* 2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control , 400 (2011).

[83] M. Sutton and A. Greene, *The art of file format fuzzing,* Blackhat USA Conference (2005).

[84] P. Godefroid, N. Klarlund, and K. Sen, *DART: directed automated random testing,* ACM Sigplan Notices , 213 (2005).

[85] P. Kahlon, *Security issues in system development life cycle of smart grid*, Ph.D. thesis, California State University, Sacramento (2011).

[86] A. J. Delozier, *Characterizations of Vulnerabilities and Countermeasures in Advanced Metering Infrastructure Collectors*, Ph.D. thesis, The Pennsylvania State University (2011).

[87] K. Paananen, *Information Security In Smart Grid Demonstration*, Ph.D. thesis, Tampere University of Technology (2011).

[88] M. Eddington, *Developing fuzzers with peach 2.0,* Proceedings of CanSecWest Applied Security Conference (2008).

[89] G. B. Frédéric Guihéry, *Auto generation of Peach pit files/fuzzers,* http://doc.netzob.org/en/latest/tutorials/peach.html, accessed: 2014-01.

[90] Nullthreat Security, *Fuzzing with Peach,* http://www.nullthreat.net/2011/01/fuzzing-with-peach-install-part-1.html, accessed: 2014-01.

[91] Ankcraz, *Peach Fuzzing Framework,* http://techank.blogspot.nl/2009/09/peach-fuzzing-framework.html, accessed: 2014-02.

[92] J. S. Dhaliwal, *Fuzz Testing,* http://www.cse.iitd.ac.in/~cs5080212/Fuzz.pdf, accessed: 2014-02.

[93] pyoor (Jason Kratzer), *Fuzzing with Peach – Part 1,* http://www.flinkd.org/2011/07/fuzzing-with-peach-part-1/ (), accessed: 2014-02.

[94] pyoor (Jason Kratzer), *Fuzzing with Peach – Part 2 (Fixups),* http://www.flinkd.org/2011/11/fuzzing-with-peach-part-2-fixups-2/ (), accessed: 2014-02.