EDSGuard: Enforcing Network Security Requirements for Energy Delivery Systems

Vu Coughlin, Carlos Rubio-Medrano and Ziming Zhao
Arizona State University
Email: {vhnguye1, crubiome, zzhao30}@asu.edu

Gail-Joon Ahn
Arizona State University and Samsung Research
Email: gahn@asu.edu

***Abstract*—Recently, *energy delivery systems* (EDS) have been targeted by sophisticated network-based attacks tailored to disrupt the proper distribution of energy among different geographical regions, resulting in non-trivial socio-economical loses and a loss of public confidence in EDS infrastructures. Such attacks were facilitated by the lack of native security measures regarding existing network communication protocols for EDS, which allowed attackers to deliberately manipulate the state of network connections between control modules and field devices. In order to address these concerns, this paper presents *EDSGuard*, a state-based firewall and monitoring tool that leverages *state-of-the-art* packet inspection techniques along with *software-defined networks* (SDN), to intelligently implement a set of security requirements and best practices for protecting EDS networks, as issued by regulatory organizations within the EDS community in the last years. In addition, *EDSGuard* implements a series of *first-response* countermeasure strategies, which can automatically react to anomalies and attacks, thus effectively mitigating their consequences and impact as a result. We provide the overall rationale behind our approach, as well as a description of our experimental results depicting a set of attack scenarios inspired by recent incidents affecting EDS infrastructures, which provide evidence of the suitability of *EDSGuard* for being fully adopted in practice.**

I. Introduction

Recently, *energy delivery systems* (EDS) [1] have gone through a modernization process that includes the introduction of intelligent field devices and software control modules, which communicate with each other through dedicated networks protocols for the purposes of automated management and control, promoting efficiency and convenience of use. Despite its inherent benefits, automation has also opened the door for non-trivial attacks targeting EDS infrastructure. Largely regarded as purely theoretical in the past, attacks on EDS infrastructure all over the world have recently become a reality. As an example, attackers successfully injected spurious network commands to disrupt EDS monitoring and control systems, thus affecting the distribution of electricity in Ukraine [2] [3], resulting in considerable organizational and financial loses and the erosion of the public's confidence in EDS infrastructure. As a response to this challenge, different EDS-related organizations have released a set of documents detailing security requirements and best practices for deterring attacks to EDS infrastructures and mitigate their consequences. As an example, the National Institute of Standards and Technology (NIST) [4], the International Electrochemical Commission [5], as well as the Energy Sector Control Systems Working Group (ESCSWG) [1] have recently published guidelines for the protection of EDS networks. Examples include the complete restriction of traffic between the *business* and the *control* subnetworks of an EDS instance, in an effort to prevent malware that may have previously infiltrated the business network from reaching devices deployed in the field.

Also recently, *software defined networks* (SDN) [6] has been widely adopted in EDS networks, due to the supported benefits and convenience for network administration, creating a new playground for researchers to explore new innovative defense solutions [7]. With that in mind, there is a need for effective implementations of the aforementioned security requirements in the context of SDN-controlled EDS networks, which can then allow for the prevention and detection of network-based attacks.

In order to address this concern, this paper presents *EDSGuard*, a firewall-like tool leveraging SDN as well as Bro [8], a well-known multi-purpose network monitor. Using these technologies as a supporting framework, our tool implements a set of security requirements for EDS allowing for the state of network communications to be constantly monitored by intelligently combining capabilities featured by both paradigms such that ongoing attacks can be detected and proper *first-response* countermeasures can be automatically deployed as a result. Overall, this paper makes the following contributions:

- We provide a summary of security requirements and best practices for protecting mission-critical EDS networks as obtained from a set of source documents, which have been in turn produced by well-reputed organizations in the EDS community.
- We also provide an approach for effectively implementing such requirements leveraging *state-of-the-art* SDN and packet inspection technologies, thus allowing for the detection of ongoing attacks to EDS infrastructure as a result.
- Finally, we also present an approach for implementing *first-response* counter-measure techniques for the attacks that are detected using our approach, thus mitigating their potential impact and consequences.

This paper is organized as follows: we start by analyzing in greater detail the problems being considered in this paper in Sec. II. Later, in Sec. III, we present *EDSGuard*, which implements our approach for solving the aforementioned problems.

We continue with a description of a series of experiments that validate the effectiveness of our tool in Sec. IV. Finally, we compare our approach with related work in Sec. V, and discuss topics for future work and conclude this paper in Sec. VI.

II. Problem Description

Recent Network-based Attacks. As discussed in Sec. I, sophisticated attacks to EDS infrastructure are no longer theoretical, as recent incidents have succeeded in disrupting the distribution of energy among geographical regions [2]. As an example, the CrashOverride [3] malware strives to automatically discover the architecture of an EDS instance, including implemented protocols and configuration settings, in an attempt to *impersonate* control hosts by distributing *forged* network packets injecting unintended commands and/or data, thus affecting the overall state of the EDS as a result. In general, in order for such attacks to succeed, they must be able to exploit communication flows within EDS networks that may have not been originally intended by administrators, along with the lack of native security measures in EDS protocols such as Modbus [9].

Security Requirements. In order to overcome these challenges, regulatory organizations within the EDS community have released several documents that describe security requirements and best practices for firewalls protecting EDS networks [4] [5] [1] [10]. Fig. 1 presents a graphical depiction of a subset of such requirements along with their corresponding source documents. As an example, both the NERC CIP Standard and the EDS Cybersecurity Procurement Language documents require EDS networks to restrict communication flows between devices by setting specific firewall rules based on IP addresses and ports for source-destination pairs.

Securing SDN Networks. As it was also stated in Sec. I, SDN is steadily making its way through full adoption in EDS networks, With that in mind, there is a need for an approach that can effectively implement firewall-based security requirements in the context of SDN-controlled EDS networks, providing protection against the attacks presented above. Concretely, the following must be supported: i) detect and react to anomalies in network packets implementing the Modbus protocol that can be potentially used for attacks, e.g., command or data injection, and, ii) enforce a well-defined set of authorized network communication flows between EDS devices, preventing the exploitation of unauthorized flows by malicious agents. In addition, the set of authorized flows must be properly enforced over time despite changes in the configuration of EDS networks as introduced by the SDN paradigm, which can potentially introduce unintended flows that can become non-trivial security vulnerabilities in the future.

Drawbacks of Existing Approaches. A naive approach for solving the problems just mentioned would include implementing the firewall-like security requirements shown in Fig. 1 by means of the manual installation of entries within the flow tables provided by network switches implementing the SDN paradigm [6]. However, such a process has the following drawbacks: i) it is error-prone as it may not support reconfigurations that introduce unintended flows into the network, thus creating security vulnerabilities as a result, and ii) such an approach does not scale as the complexity grows along with the size, e.g., number of hosts and possible communication flows, of the EDS network. Moreover, existing firewall solutions for SDN [11] [12] may be able to avoid such drawbacks by handling the update of switch-based flow entries in an automated way. However, such approaches may not be able to maintain a consistency between policies defined at the firewall level and the changes in the EDS network configuration at the switch level, which can potentially introduce unintended flows that can be later exploited by malicious agents. Finally, these approaches provide no native support for security measures for EDS protocols, thus opening the door for the attacks just discussed.

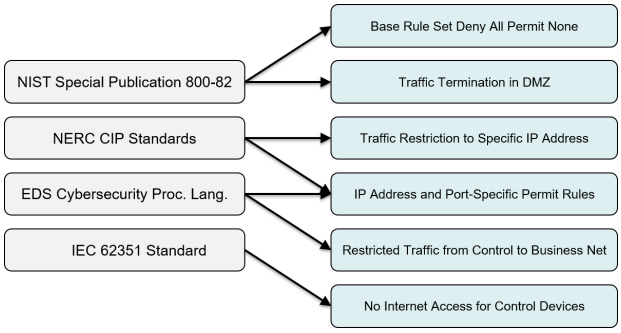


Figure 1: A set of requirements for firewall applications for protecting EDS networks, as provided by different sources within the EDS community.

III. *EDSGuard*: Proactive Monitoring and Response for SDN-based EDS Networks

In order to effectively respond to the challenges just presented, we introduce *EDSGuard*, an SDN-based firewall application implementing network security requirements for EDS. *EDSGuard* leverages the features offered by recent SDN implementations, in which network packets can be forwarded to the SDN controller [13], implementing custom-made applications for further inspection before they are allowed back into the network. Such a feature, which has been also explored by previous approaches in the literature [11], is then leveraged in the context of EDS networks for: i) inspecting all packets that enter the network, including those creating new communication flows between EDS devices and, ii) implementing security-based packet-inspection capabilities that overcome the current limitations of EDS protocols.

With this in mind, we now present the supporting data structures as well as the set of *first-response* countermeasure strategies implemented by *EDSGuard*. Later, using such discussion a basis, we provide an extended description on the way *EDSGuard* prevents ongoing and future attacks to SDN-controlled EDS networks, including how the security

requirements presented in Sec. II are effectively implemented by our tool.

High-Level Firewall Policy. *EDSGuard* models the firewall-like security requirements shown in Fig. 1 by implementing a *high-level* firewall policy that contains rules restricting the set of authorized communication flows within an EDS network. As an example, a policy rule may restrict any traffic from the *Control* to the *Business* subnetworks and vice versa, thus preventing compromised hosts within the Business network from reaching EDS devices in the field. In such a context, network-specific information required for the specification of *high-level* policies, e.g., IP Addresses and ports, may be defined beforehand by network administrators, following the paradigm commonly used for firewall applications in practice. Also, auxiliary tools may be used for automated policy creation [14]. Fig. 3 shows the schema for defining rules within the *high-level* policy defined by *EDSGuard*.

First-response Resolution Strategies. As introduced in Sec. I, *EDSGuard* implements a set of *first-response* strategies that go beyond detection to proactively react and reduce the impact and consequences of ongoing and future attacks to EDS networks. Such strategies include:

1) *Flow Rejecting*. As implied by its name, this strategy rejects the installation of new data flows in the tables implemented by the SDN-controlled switches. As it will be discussed later, it can be activated in cases *EDSGuard* detects a violation of its *high-level* firewall policy.
2) *Flow Removing*. This strategy is intended to maintain a consistency between the *high-level* firewall policy and the flow tables implemented by the switches, in order to prevent the existence of unintended flows that can lead to security vulnerabilities. There are two possible cases to apply this strategy: (1) When a *high-level* policy rule is removed, e.g., as a result of a network reconfiguration, every flow rule associated with it will be removed from the switch tables. (2) In addition, *EDSGuard* supports a priority scheme for the introduction of rules with different levels of *priority*, represented by means of a numerical ordering, as shown in Fig. 3, allowing for rules with a high priority to take precedence at policy evaluation time. In such a case, when a *higher* priority firewall rule is added to the policy overwriting a previously-existing *lower* one, every flow associated with the lower priority rule will be removed from the switch tables.
3) *Update Rejecting*. Going in the opposite direction, this strategy rejects any changes in the flow tables that may ultimately result in a violation of the *high-level* firewall policy. Such changes may include adding, editing and deleting table entries that may result in changes in the communication flows available in the network.
4) *Packet Blocking*. Finally, this strategy blocks an anomalous network packet by instructing switches to drop it from the network, keeping log records for future analysis.

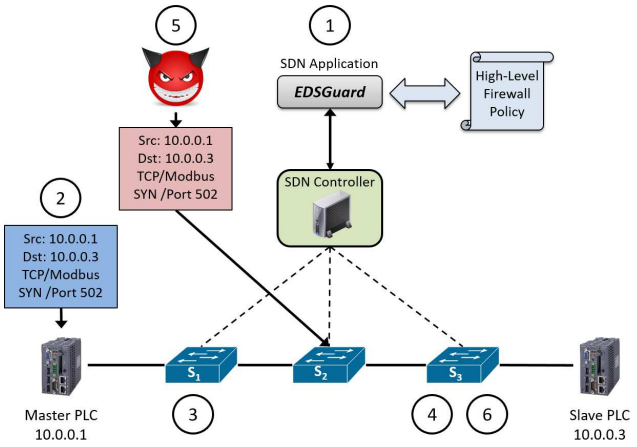


Figure 2: *EDSGuard* handling an attack on an EDS network: initially, *EDSGuard* is installed as an SDN application (1). Later, the *master* PLC initiates communication with a *slave* PLC by sending a network packet (2), which is then intercepted by the *ingress* switch and forwarded to *EDSGuard* (3). Once the aforementioned packet reaches the *egress* switch, it is forwarded back to *EDSGuard* for processing (4). Later, an attacker may try to inject a specially-crafted packet impersonating the *master* PLC (5), which is then detected by *EDSGuard* and rejected at the *egress* switch (6).

```
{ "fw_rule": {
"ruleId": <unique number>,
"sourceIP": <source IP address>,
"destIP": <destination IP address>,
"protocol": <UDP / TCP>,
"srcPort": <between 1 and 65535>,
"dstPort": <between 1 and 65535>,
"priority": <low= 1, high= 65535>,
"action": <DENY / ALLOW>
}}
```

Figure 3: A *high-level* policy rule encoded as a JSON object.

Detection and Response to Ongoing Attacks. Leveraging its *high-level* firewall policy along with the SDN packet-inspection capabilities and the *first-response* strategies just discussed above, *EDSGuard* provides support for detecting and responding to ongoing attacks implementing an approach that is graphically depicted in Fig. 4, which can be further described as follows:

1) *Violation of the High-Level Policy. EDSGuard* rejects any packets for a network flow in direct violation of the *high-level* policy. Such a feature is graphically depicted in Fig. 2: a new packet arriving at an SDN-enabled switch, which happens to create a new data flow, is forwarded to the SDN controller and ultimately to *EDSGuard*, which then retrieves the *high-level* firewall policy and checks for violations. If no violations are found, the packet is then allowed to continue through the network. Violating packets are then the subject of the *Flow Rejecting* and the *Packet Blocking* first-response strategies. Conversely, network packets not creating a

new flow are then analyzed by an inspection engine leveraging the Bro framework for anomalies, as it is described next.

2) *Security-based Packet Inspection.*
EDSGuard also implements a dedicated engine that leverages the Bro analysis capabilities to provide *in-depth* inspection of network packets depicting the Modbus and the *address resolution protocol* (ARP) protocols. In general, such an engine supports the following:
Communication Flow Checking. Our engine inspects Modbus packets to verify the communication flow between EDS devices, detecting anomalies in the direction of packets as well as in their inner contents. As an example, referring back to Fig. 2, a *slave* PLC may be only allowed to send packets to a *master* PLC containing a response to a given data access request. In a normal communication flow, a *master* PLC initiates the session to the *slave* PLC. However, if a *slave* PLC starts communication to its *master* or other *slave* PLC(s), then this is considered as an anomaly and an alert will be raised.
DoS Prevention. Following a similar scheme, our engine can also prevent *denial of service* (DoS) attacks by employing the capabilities offered by Bro for detecting unusual amounts of traffic originating from/to a given device within the network, raising an alert when such amounts exceed a predefined threshold.
ARP Packet Inspection. Our engine also inspects packets implementing the ARP protocol, following an approach inspired by the one presented by Udd et al. [15]. Such a feature is implementing by keeping an ARP *whitelist*, consisting of pairs of matching IP address and Media Access Control (MAC) addresses. Then, the engine leverages the Bro ARP analysis capabilities [16] to generate alerts whenever there exists an anomaly in an ARP packet under inspection. As an example, a dedicated attack may try to inject ARP packets such that a communication flow between a compromised host and an EDS field device is created, thus opening the door for potential attacks. When the alert is generated, our engine compares the faulting pair against the aforementioned whitelist to determine its originating source within the network.
Finally, alerts raised by our engine are ultimately forwarded to *EDSGuard*, which then applies the *Packet Blocking* first-response strategy, as shown in Fig. 4.

Prevention of Potential Security Vulnerabilities. In addition, *EDSGuard* can also prevent future security vulnerabilities that arise from reconfigurations of the EDS network, e.g., the introduction of new communication flows between devices as a response to physical changes/emergencies within the EDS infrastructure. For such a purpose, *EDSGuard* implements an strategy comparing the requested changes against the *high-level* firewall policy and the flow table entries implemented at the switch level, following an approach inspired by the techniques proposed in [11].

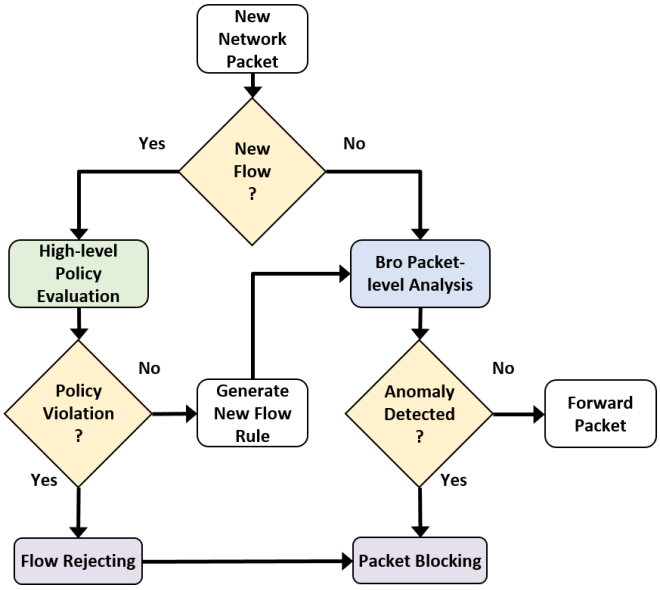


Figure 4: The *EDSGuard* violation and anomaly detection mechanism to deter ongoing attacks to EDS networks. Incoming packets are compared against the *high-level* policy and the packet-level inspection engine implemented by means of the Bro framework. Faulting packets are preventing from generating new communication flows and are ultimately dropped from the network.

1) *High-Level Policy Consistency. EDSGuard* maintains the consistency between the *high-level* firewall policy and the flow tables implemented by SDN switches by translating the policy rules into entries for the flow tables. Then, each time a new network flow is introduced as a result of a reconfiguration, consistency with the *high-level* policy is checked, dropping new flows that happen to violate a rule in such a policy and preventing any modifications to the flow table entries, thus preventing the introduction of potential security vulnerabilities. This functionality is ultimately achieved by means of the *Flow Removing* first-response strategy.
2) *Switch-based Flow Table Consistency.* Conversely, every time the flow table entries at switches are reconfigured, *EDSGuard* inspects the *high-level* policy and rejects any changes that may introduce flows not authorized by such a policy, thus effectively implementing the *Update Rejecting* first-response strategy.

Implementing Security Requirements. Finally, we leverage the features offered by *EDSGuard* to implement the security requirements shown in Fig. 1 as follows: for requirements restricting network traffic between hosts in the network, e.g., the rules restricting traffic between specific IP addresses and TCP/UDP ports, and the rules restricting access between EDS sub-networks and the Internet, we can deploy a set of rules for the *high-level* policy implemented by our approach. In addition, information about the EDS network architecture can be used to populate the ARP whitelist implemented by our Bro-based packet-inspection engine, thus providing extended

monitoring capabilities for effectively enforcing the traffic restrictions just discussed. Moreover, the security requirement defining the need for network traffic to be terminated within a *demilitarized zone* (DMZ) can be implemented by installing two *high-level* firewall rules allow the communication between the different subnetworks and the DMZ in separate ways, preventing the direct connection between the former two as devised by the other security requirement discussed before. Finally, the security requirement defining the need for a base *deny-all* firewall rule can be implemented by means of a default rule for the *high-level* firewall policy implemented by *EDSGuard*.

IV. Experimental Evaluation

In order to provide evidence of the suitability of *EDSGuard* for being deployed in practice, we have created a simulation testbed that resembles existing EDS infrastructures commonly found in practice. In addition, we have equipped our testbed with a communications network that implements the SDN paradigm for administration and control. Using this testbed, we then performed a series of experiments depicting case scenarios based on real-life attacks that have been recently reported. For each experiment, we present its motivation, rationale, and the way our *EDSGuard* approach can effective handle it.

Experimental Testbed. As an initial step towards providing a realistic testbed for our approach, we resorted to a simulation infrastructure that follows the architectural directives depicted in the National Institute of Standards and Technology (NIST) publication 800-82 revision 2 [4]. Such an architecture, graphically depicted in Fig. 5, comprises several hosts organized into three different sub-networks: the *Control* sub-network, the *Business* network, and *demilitarized zone* (DMZ). We performed our experiments on a 48-core server that uses both OpenStack [17] and Mininet [18] Also, SDN functionality was implemented by means of the OpenDaylight (ODL) controller [13] and Openvswitch [19]. Our SDN controller, as well as the rest of our simulated EDS infrastructure, were instantiated inside dedicated virtual machines running in our cloud, allowing for all the network communication between them, e.g., by means of the Modbus protocol, to be actively collected for experimental purposes. We also implemented our own simulation software for the EDS field devices depicted in Fig. 5. Finally, we deployed *EDSGuard* by assuming an initial *high-level* policy firewall implementing the security requirements discussed in Sec. III taking into account the architecture depicted in Fig. 5. As an example, paraphrasing Fig. 1, communications between the *Business* and the *Control* sub-networks were forbidden by creating rules comprising devices located on each sub-network.

Adversarial Models. Following the approach described in Sec. III, an adversary attempting to disrupt an EDS network implementing our proposed *EDSGuard* tool must bypass two layers of security. First, he/she must overcome the SDN-based firewall that mediates data flows between hosts in the network. Second, an adversary that is able to bypass such a firewall,

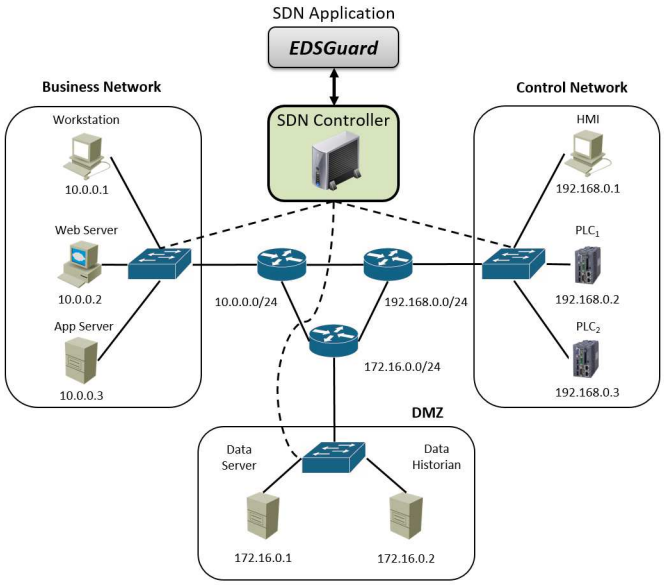
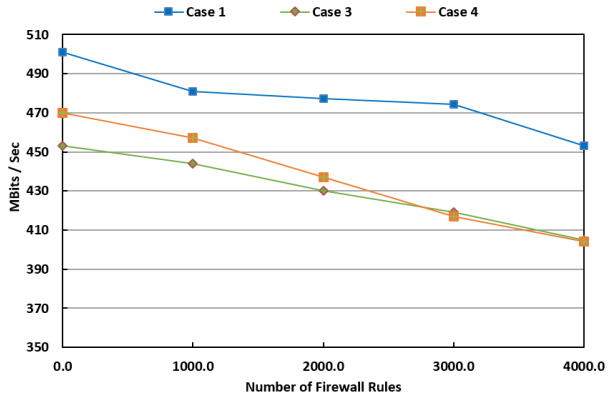


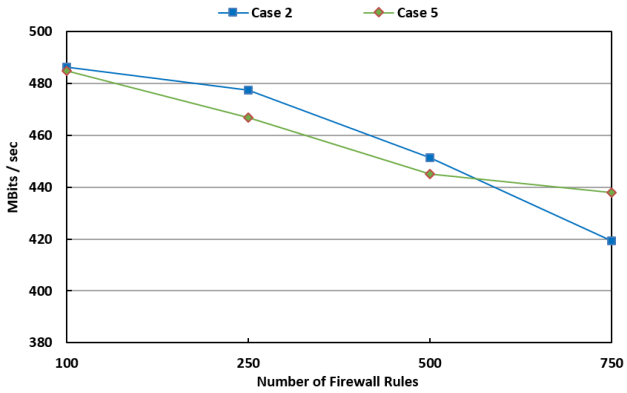
Figure 5: A simulated EDS infrastructure for experimental purposes based on a description from NIST [4]: the *Business* and *Control* sub-networks are separated by means of a *demilitarized zone* (DMZ). Our *EDSGuard* tool is implemented as an SDN application running on top of an SDN controller.

and has been able to install a specialized malware inside the network, must then bypass the protection offered by our packet-inspection engine leveraging the Bro framework. In this context, a *compromised host* attack occurs when an adversary is able to install dedicated malware inside an EDS field device, which, besides disrupting the device's functionality, tries to spread itself to other devices within its reach. When a *master* PLC is compromised, it can generate messages to *slave* PLCs, potentially causing them to malfunction, affecting the overall EDS infrastructure as a result. Conversely, a *network-base* attack occurs when an adversary is able to compromise an EDS device for the purposes of eavesdropping in the network to collect sensitive information. Additionally, since no encryption is used to secure network packets implementing the Modbus protocol [20], an adversary can also modify them as they go through the network and effectively perform a *man-in-the-middle* attack.

Case Scenario 1: Flow Rejection. Following the adversarial models just described, our first experiment modeled an scenario in which an attacker tries to infiltrate an EDS network by first attempting to install a new data flow on a given SDN-enabled switch, thus ultimately establishing communications with a target EDS device or control host. Following the experimental testbed described in Fig. 5, we crafted an automated script that targets the switch located inside the *Control* network from a host physically located inside the *Business* network, thus depicting an scenario in which an attacker has been able to compromise a host in such a network in the first place and then attempts to communicate with EDS devices located



(a) Case Scenarios 1, 3, 4.



(b) Case Scenarios 2, 5.

Figure 6: Performance Evaluation for Different Case Scenarios.

in the *Control* network. Our *EDSGuard* tool prevented this attack scenario by detecting the requested change on the flow tables of the aforementioned switch and cross-checking it with its *high-level* policy, thus concluding that the installation of such a data flow causes a policy violation. As a result, the *flow rejection* resolution strategy was invoked, ultimately instructing the SDN controller to reject any changes to existing flow tables at the switch level.

The experimental results for all the case scenarios included in this Section are shown in Fig 6 (a) and Fig. 6 (b). In our experiments, we measured the runtime performance of *EDSGuard* by using the `iPerf` [21] tool to calculate the latency that *EDSGuard* introduced into the network. We also varied the number of rules contained within the *high-level* policy implemented by our approach. Finally, in the experimental results for the case scenario 1 just described, increasing the number of such rules (and hence the number of flow table rules managed by the SDN switches), causes the overall network bandwidth to decrease within manageable means, as shown in Fig. 6 (a).

Case Scenario 2: Flow Removing. In a subsequent experiment, we explored an scenario in which an attacker tries to remove an existing flow rule located inside a SDN-enabled switch, thus effectively disrupting the communication between EDS devices, which may ultimately affect the proper distribution of energy, as control hosts may incorrectly assume the targeted field devices are malfunctioning due to the lack of data communications with them. Referring back to Fig. 5, such an attack was materialized by targeting the switch inside the *Control* network, which may then prevent a given EDS field device, e.g., the PLC located at 192.168.0.3, from communicating with its corresponding HMI control host located at 192.168.0.1. As in our previous scenario, such a change was rejected by *EDSGuard* as it was regarded as a *high-level* policy violation, and the *flow removal* resolution strategy was ultimately used. In the experimental results shown in Fig. 6 (b), the number of *high-level* firewall rules that are related to the targeted flow rule has a direct impact in the overall network latency, since *EDSGuard* needs to process all of them to ensure the attempted removal of the flow rule will not cause any security violations.

Case Scenario 3: Packet Anomaly. In another experiment, we simulated an scenario in which an attacker triggers a DoS attack on an EDS field device, thus effectively disrupting its normal functionality with similar consequences as the ones discussed before. Resorting back to Fig. 5, this experiment involved an attacker leveraging an already-compromised host within the *Business* network (10.0.0.1) to overflow communications with an EDS field device located within the *Control* network (192.168.0.2). *EDSGuard* was able to successfully detect such an anomaly by means of its packet-inspection engine, which in turn leverages the Bro monitoring framework for analyzing abnormal traffic flows, as presented in Sec. III. As a result of the alerts being raised, *EDSGuard* was able to apply the *packet blocking* resolution strategy, thus preventing the target EDS device from receiving any undesired traffic at all. As expected, in the experiments related to this case scenario, the use of the packet-inspection engine causes additional overhead on the network, as shown in Fig. 6 (a).

Case Scenario 4: Packet Integrity. One downside of the SDN controller is that once the network topology is established and the flow rules are installed, the controller does not have knowledge of what is happening within a network. As a result, a *man-in-the-middle* attack can be successfully performed. In Figs. 5, an adversary can be located at PLC_1 listening to the conversation between HMI and PLC_2. In this experiment, *EDSGuard* was able to successfully detect such an attack by utilizing the *ARP list* mentioned in Section III. When our packet-inspection engine detects that the packet does not match the ARP list, an alert is raised. Then, a firewall rule is constructed based on this alert to prevent further communication between the attacker (192.168.0.2) and its victims (192.168.0.1 and 192.168.0.3). Similar to case 3, the introduction of our packet-inspection engine has a direct effect in the overall performance of the network, as shown in Fig. 6 (a).

Case Scenario 5: Legit Infrastructure Changes. Finally, we explored an scenario in which a new data flow between two EDS field devices was established by EDS operators as a response to physical events that demanded a design of the strategy for energy distribution. Our experiment then involved the two PLCs shown in Fig. 5 being able to communicate with each other as it is also depicted in Figs. 2. As a response, our *EDSGuard* deployed the *update-rejecting* strategy as the aforementioned operation was deemed as a violation of the *high-level* firewall policy. Such a resolution strategy may indeed prevent EDS operators from inadvertently introducing data flows that may become security vulnerabilities in the future, forcing them to carefully revise the *high-level* firewall policy implemented by *EDSGuard* before any changes in the network can be deployed. Finally, as with the previous case scenario 2, the number of *relevant* rules contained within the *high-level* policy has a direct effect on the overall network latency, as shown in Fig. 6 (b).

V. Related Work

Over the past few years, SDN has gained enormous popularity by allowing enterprises to logically separate the control and data planes of a network device, thus making data networks more cost-effectively as a result. In such a context, firewall-like applications for SDN [22] [12] resemble traditional firewalls as they operate on a centralized approach for the monitoring of the network, implementing a centralized policy for administering allowed or denied data flows between devices. In a similar fashion, Bro has been leveraged in the literature to provide monitoring techniques for SDN networks [23], in an effort to enhance the capabilities exhibited by traditional firewalls by detecting anomalies based on packet-inspection techniques. While serving as an initial inspiration for our *EDSGuard* approach, the aforementioned approaches provide no native support for implementing security requirements for EDS networks, e.g., the implementation of traffic restrictions between different EDS control hosts. In addition, they lack any capabilities for implementing countermeasure techniques once a given network anomaly has been detected. As discussed in Sec. III, *EDSGuard* provides support for implementing *first-response* countermeasures to ongoing attacks, thus reducing the response time to incidents, and potentially damages to the overall EDS infrastructure as a result.

VI. Conclusions and Future Work

In this paper, we have presented *EDSGuard*, a dedicated tool that implements a set of well-defined security requirements for preventing, detecting and reacting to network-based attacks to EDS infrastructures. In addition, we have shown how *EDSGuard* can effectively handle a set of case scenarios based on real-life attacks. As of today, we are actively working towards providing an enhanced experimental testbed, which includes additional case scenarios as well as extended simulations of EDS infrastructure, such that we can obtain additional evidence of the suitability of our approach for being successfully deployed in practice.

Acknowledgments and Disclaimer

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000780 and by a grant from the Center for Cybersecurity and Digital Forensics at Arizona State University. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of United States Government or any agency thereof.

References

[1] Energy Sector Control Systems Working Group (ESCSWG), "Cybersecurity Procurement Language for Energy Delivery Systems," April 2014. [Online]. Available: https://www.energy.gov/oe/downloads/cybersecurity-procurement-language-energy-delivery-april-2014

[2] D. E. Whitehead, K. Owens, D. Gammel, and J. Smith, "Ukraine cyber-induced power outage: Analysis and practical mitigation strategies," in *70th Annual Conf. for Protective Relay Engineers*. IEEE, 2017.

[3] Dragos, Inc., "CRASHOVERRIDE: Analysis of the Threat to Electric Grid Operations," https://dragos.com/blog/crashoverride/, Dragos, Inc., Tech. Rep., 06 2017.

[4] NIST, "NIST Special Publication 800-82 Revision 2 Guide to Industrial Control Systems (ICS) Security," May 2015. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf

[5] International Electrochemical Commission, "IEC TC57 WG15: IEC 63251 Security Standards for the Power System Information Infrastructure," June 2012. [Online]. Available: http://www.iec.ch/smartgrid/standards/

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '07)*. ACM, 2007, pp. 1–12.

[7] D. Jin, Z. Li, C. Hannon, C. Chen, J. Wang, M. Shahidehpour, and C. W. Lee, "Toward a cyber resilient and secure microgrid using software-defined networking," *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2494–2504, Sept 2017.

[8] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[9] "Modbus." [Online]. Available: https://goo.gl/L1w8Jy

[10] NERC, "CIP Standards," 2017. [Online]. Available: www.nerc.com/pa/Stand/Pages/CIPStandards.aspx

[11] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "FLOWGUARD: building robust firewalls for software-defined networks," in *Third Workshop on Hot Topics in Software Defined Networking*. ACM, 2014, pp. 97–102.

[12] S. Zerkane, D. Espes, P. Le Parc, and F. Cuppens, "Software defined networking reactive stateful firewall," in *ICT Systems Security and Privacy Protection*. Springer, 2016, pp. 119–132.

[13] "Opendaylight." [Online]. Available: https://www.opendaylight.org/

[14] E. Saboori, S. Parsazad, and Y. Sanatkhani, "Automatic firewall rules generator for anomaly detection systems with apriori algorithm," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 6, Aug 2010, pp. V6–57–V6–60.

[15] R. Udd, M. Asplund, S. Nadjm-Tehrani, M. Kazemtabrizi, and M. Ekstedt, "Exploiting Bro for Intrusion Detection in a SCADA System," in *Int. Workshop on Cyber-Physical System Sec. (CPSS '16)*. ACM, 2016, pp. 44–51.

[16] P. Erickson, "http://mailman.icsi.berkeley.edu/pipermail/bro/2011-november/004415.html."

[17] "Openstack." [Online]. Available: https://www.openstack.org/

[18] "Mininet." [Online]. Available: http://mininet.org/

[19] "Openvswitch." [Online]. Available: http://www.openvswitch.org/

[20] A. Swales *et al.*, "Open modbus/tcp specification," *Schneider Electric*, vol. 29, 1999.

[21] "iPerf." [Online]. Available: https://iperf.fr/

[22] H. Hu, G.-J. Ahn, W. Han, and Z. Zhao, "Towards a reliable SDN firewall," in *Presented as part of the Open Networking Summit 2014 (ONS 2014)*. Santa Clara, CA: USENIX, 2014.

[23] R. Koning, N. Buraglio, C. de Laat, and P. Grosso, "Coreflow: Enriching bro security events using network traffic monitoring data," *Future Generation Computer Systems*, vol. 79, pp. 235 – 242, 2018.