# Automated Incident Response for Industrial Control Systems Leveraging Software-defined Networking

Florian Patzer[1], Ankush Meshram[2] and Maximilian Heß[1]
[1]*Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB), Karlsruhe, Germany*
[2]*Vision and Fusion Laboratory (IES), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

Keywords: Incident Response, SDN Security, Industrial Control Systems, ICS Security, Software-defined Networking.

Abstract: Modern technologies and concepts for Industrial Control Systems (ICS) are evolving towards high flexibility of processes and respectively networks. Such dynamic networks are already functioning well, for example in data centres. This is enabled by application of the Software-defined Networking (SDN) paradigm. For this reason, ICS is currently adopting SDN. The concept of having a centralized view of the network and generating packet forwarding rules to control it enables performing automated responses to network events and classified incidents via SDN. This automation can provide timely and, due to the holistic view of the network, accurate incident response actions. However, availability, safety, real-time and redundancy requirements within the ICS domain restrict the application of such an automated approach. At present, SDN-based incident response (SDN-IR) does not take into consideration these requirements.
In this work, we identify possible SND-based response actions to ICS incidents and introduce classification of assets and links. Furthermore, we present a concept for SDN-IR where a predefined rule set restricts the response actions based on the asset's classification thereby satisfying ICS specific requirements. Subsequently, we describe and evaluate a prototype implementation of this concept, built with the open-source SDN platform OpenDaylight and the SDN protocol OpenFlow.



## 1 INTRODUCTION

Highly flexible industrial processes are currently evolving and spreading rapidly, due to new possibilities digitalization provides. Hence, modern Industrial Control Systems (*ICS*) face the need for highly flexible networks. Such dynamic networks are already well established in other domains, like data centres. Software-defined Networking (*SDN*) is the enabling paradigm in these domains and currently adopted by ICS in order to meet the need for dynamic networks. SDN is highly efficient in providing flexible centralized management of data flows. This is achieved by separating the control plane from the data plane (Kim and Feamster, 2013). The control plane has a logically centralized view of the network and holds detailed knowledge of its topology. The SDN controller utilizes this knowledge to calculate packet forwarding rules and deploys them on the switches. For this, the forwarding rule gets added to a *flow table*, which is the reason why such rules are often called *flow entries*. The switches follow the deployed flow entries for each packet matching one of them. If a packet cannot be matched to a flow entry, the switch sends this packet to the SDN controller, which then can react e.g. by generating new flow entries.Since SDN management (i.e. the controller and its environment) control the data flow within the whole SDN realm, automated response to events and classified incidents can be realized. As an example, (Martins and Campos, 2016) presented the automated incident response with SDN using a security architecture where the Intrusion Detection System (IDS) Snort sends alerts to an SDN controller. Based on this alert, corrupt devices are isolated and blacklisted via certain flow entries. This enables a fast reaction to detected events. The importance of such fast responses can be comprehended from reading the SANS Incident Response Survey 2017 (Bromiley, 2017). One can clearly conclude from the survey that faster containment and short-term remediation help to prevent attack propagation and aid damage regulation. Moreover, in ICS, attacks tend to have dangerous impact on the safety of humans and quickly cause massive financial damage. Thus, in such systems containment and short-term remediation in a matter of milliseconds are of special interest. Applying SDN for automated incident response can achieve this objective.



Patzer, F., Meshram, A. and Heß, M.
Automated Incident Response for Industrial Control Systems Leveraging Software-defined Networking.
DOI: 10.5220/0007359503190327
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 319-327
ISBN: 978-989-758-359-9

Leveraging SDN for incident response is still a very young research field. However, first research results already exist. The authors of (Koulouris, T., Casassa Mont, M. and Arnell, S., 2017) introduced *SDN4S* (SDN for Security), an SDN-based incident response (*SDN-IR*) system leveraging automated countermeasures to minimize remediation time. The authors introduced the concept of playbooks, consisting of a set of triggers (corresponding to the incident addressed by the playbook) and a set of executable actions (building the response strategy). Therefore, playbooks describe the decisions an SDN controller has to make upon the arrival of an incident alert. Since their concept is not designed for ICS environments, it does not meet ICS-specific requirements like availability, reliability, safety, time sensitivity and redundancy. We return to these requirements in section 2.2. The requirements restrict the application of SDN-IR within the ICS domain.

Even research results addressing ICS-specific SDN-IR are available. (Piedrahita et al., 2018) shows how SDN and Network Function Virtualization (NFV) can be used for automated incident response in ICS. The authors propose virtual incident response functions which replace ICS components under attack. If an attack on SCADA level is identified, the virtual representation is used to create a honeypot of the control system. (Di Lallo et al., 2017) addresses the real-time requirements for SDN-based ICS security. In their approach they leverage spare of bandwidths to transfer replicas of the sent packets within the network to an IDS. This shows how SDN can assist in the detection phase of incident response without endangering the ICS network's quality of service guarantees.

At present, to the best of our knowledge no SDN-IR concept is able to support all the peculiarities of the ICS domain. This issue prevents the application of SDN-IR in SDN-based industrial networks. However, we found SDN4S to be a simple and useful bases to build an ICS-capable SDN-IR solution. Moreover, the architecture of SDN4S allowed us to extend it to incorporate ICS-requirements. For this, we extended the playbook-based concept of SDN4S by introducing classification of assets (hosts, network components and links) which represent ICS-specific requirements and restrictive rules. Thus, as in SDN4S, the playbook defines a set of response actions to perform for a given incident/asset type pair. Afterwards the rules matching the response actions, asset type and the asset's classification are used to derive preconditions which have to be met, before the actions are allowed to be executed. If all applying rules allow the performing of an action, it is transformed into flow entries for reconfiguration of the relevant SDN switches. To evaluate the feasibility of the concept's implementation with common SDN platforms, we present a prototype implementation built upon the open-source SDN platform OpenDaylight and the SDN protocol OpenFlow. Afterwards, we describe some of the results of the evaluation which was performed by applying the prototype to example topologies using the Mininet network emulator.

# 2 SDN-IR FOR ICS CONCEPT

In this chapter our concept of ICS-capable SDN-IR is explained. The core concept of our approach can be summarized as follows: A playbook entry is a mapping from an incident-asset type pair to a list of response actions. For each asset type-response action pair, certain rules apply for certain classes. These rules restrict the conducting of the response actions by defining respective preconditions.
For this, we first describe the types of assets and incidents we considered. Afterwards, we introduce basic classifications of assets, followed by the definition of a basic rule set. At the end of this chapter we present how these definitions are used in an SDN architecture. The here described SDN-IR concept was developed for alerts which are either initiated by host-based and network-based intrusion detection systems or manually triggered, e.g. by an incident responder for the purpose of activating intensive monitoring or isolation of a certain component. Since this allows a wide heterogeneity of incident types being handled, we abstractly classify the incidents covered by our approach as *compromised host*, *compromised network component* and *malicious link*, where hosts are network participants and a link describes any identifiable end-to-end connection (even if that connection is not currently established). Furthermore, when we want to refer to hosts and switches collectively, we use the term *node*.

## 2.1 Response Actions

The incident response process can be divided into seven phases: Preparation, Detection, Analysis, Containment, Eradication, Recovery and Post Incident Activity (Cichonski et al., 2012). The actions we identified for SDN-IR primarily support the Analysis and Containment phases and are described in the following paragraphs.

**Isolate Host.** Hosts can be isolated from the rest of the network, if the incident suggests that they have

been compromised. This can be achieved by deploying high-priority flow entries to the SDN switch the device is directly connected to. Following the flow entries, the switch drops all packages from and to this device.

Another strategy is the isolation of linked partners from the rest of the network, but not from each other. More generally, if a host is potentially compromised, it might be necessary to isolate it from the rest of the network. However, if it has a functionally critical link to another host, it seems to be a valid strategy to keep that link open. Since flow entries can be defined e.g. for specific TCP/IP connections, this can be achieved by deploying high priority flow entries allowing the critical link and lower priority flow entries dropping all packets from and to the host in question.

**Isolate Switch.** Like hosts, SDN switches can also be isolated from the rest of the network. Such an incident response action however has significantly more impact on the rest of the network than isolating a host. Isolating switches can result in isolating other nodes as well (cf. topology in figure 3 where isolation of switch *openflow:1* results in isolation of *host3* and switch *openflow:6*). In general, if any link cannot be redirected via paths not containing the switch in question, the availability of, for example, services is being affected negatively. Thus, several conditions have to be checked before isolating a switch (cf. section 2.3).

**Block Links.** Blocking certain links might be an especially interesting approach against Denial-of-Service attacks. For example, if a node within the network is affecting others using a Denial-of-Service attack, the links used for this attack can often be identified quickly and can then be blocked without isolating the whole node which might be crucial for the system. This example shows, blockage of links can be chosen as a lighter alternative to the isolation of whole nodes.

**Mirroring/Packet Replicas.** The arguably most practical response is the replication of packets, e.g. for monitoring. Whereas in traditional networking special switches (or network TAPs) with the ability to perform mirroring were needed, SDN can conduct mirroring on every SDN switch by deploying respective flow entries. This response action creates additional traffic, but does not affect the network beyond that. As already mentioned, (Di Lallo et al., 2017) proposed an approach to minimize this effect.
If the replication of packets is performed to monitor a host, the replication point or points can be chosen by selecting all SDN switches the host is directly connected to. If the monitoring of an SDN switch is desired, the objective is generally a maximization of the amount of monitored links directed over that switch. For links between communication partners directly connected to that switch, trustworthy monitoring is only possible, if the switch to monitor is not a suspect of compromising and can therefore replicate the packets itself. Other links can be replicated by other SDN switches along their paths.

**Virtualization.** As proposed by (Piedrahita et al., 2018) it is possible to support a system under attack with several virtualization strategies. For once it might be possible to replace a host with its virtual representation. This can even be extended towards replacing physical processes with respective simulations. The virtualization approach can therefore be used to considerably improve the resilience of a system. Even though this strategy is very promising and worth more research, its complexity and heterogeneity disqualified it as part of our current research. However, the playbook approach we follow is sufficient to trigger such incident response actions as well.

**Notification.** The SDN-IR solution can create notifications which will be sent to configured contact points, like an administrator's email. The notification action can be beneficial in several states of the SDN-IR progress. It can be sent directly after an alert is received, to inform the administrator and provide first reaction suggestions which, once permission is granted, can be performed automatically. Furthermore, the notification can be enriched with important knowledge about the current network layout (or topology) in order to support analysis and reduce reaction time. In addition, the notification can provide a report about the actions taken during an automated incident response. As already motivated in the introduction (section 1), the here described response actions have to be restricted for certain classes of assets. These classes are explained in the following section.

## 2.2 Classification of Assets

In this section, we introduce some basic classes of hosts, network components and links. In various environments, it might be necessary to add more classes or adapt the here defined basic classes.

Certain nodes and links are critically important e.g for controlling the production plant or to perform safety functions. Typically, the data transmission between hosts or via links may not be interrupted, not even for containment. Such nodes and

critical communication links are henceforth defined as *functionally-critical*.

Time sensitivity is an important requirement within ICS. Certain links must meet specified time conditions and adhere to them. These links are defined as *time-critical*.

Redundant network paths are incorporated in ICS for two reasons: to increase the probability that, in case of attack or disturbance, one of the paths remains functional; and to increase total bandwidth power by adding additional paths. For such redundant paths, it is always important that the path contains a disjoint set of physical transmission nodes. Hence redirecting or interrupting redundant links might be undesired. For this work, links instead of paths are classified as *redundant*. If a link is classified as *redundant* more than one path has to satisfy this link and the paths have to be physically disjoint. Furthermore, hosts and network components might also be classified as *redundant*.

Typically links can belong to more than one class. For example, *functionally-critical* and *time-critical* links are common for safety systems.

## 2.3 Restrictive Rules

In the introduction, we already motivated why ICS requirements have to be considered during the selection of applicable response actions from playbooks. To enforce this consideration, a rule set has to be available for the SDN-IR instance. We define this set as restrictive, adjustable preconditions for response actions taking into account ICS requirements. The ICS requirements are expressed by the classes described in section 2.2. We explain each rule and define informal expressions wherever reasonable to support a better understanding. The expressions are composed of boolean statements. If the statements result in the value true the respective action can be performed and vice versa. We chose this additional representation of rules to provide a valuable template for implementations and extensions. The here presented rules are neither complete nor applicable for every possible system. They only provide a starting point to implement own specialized rule sets.

For the mentioned informal expressions the following definitions are needed:

- $x \in \mathbb{N}$ describes an index enumerating network participants
- $host_x$, $mac_x$, $ip_x$ and $port_x$ represents a host, MAC address, IP address and TCP/UDP port
- $link_{i,j} := \{host_i, host_j, mac_i, mac_j, ip_i, ip_j, port_i, port_j\}$ represents a link between $host_i$ and $host_j$
- $L$ represents the set of known links
- $path_{i,j}$ represents a path from $host_i$ to $host_j$ as an ordered set of switches
- $\mathsf{time\_crit}_\tau(link)$ states that $link$ is marked as *time-critical* with a latency threshold of $\tau$
- $\mathsf{meets}_\tau(path)$ states that $path$ can guarantee a latency beneath $\tau$
- $path_{link_k}$ represents the path of $link_k$

In production the definition of $link_{i,j}$ would generally contain too much detail, since the ports (and eventually IP address) are not static. Thus, they have to be replaced, e.g. by customized categories like *"OPC UA link between* $(host_i, ip_i)$ *and* $(host_j, ip_j)$*"*.
The following paragraphs describe our exemplary rule set.

**Blocking a Link.** Links should not be blocked, if they are classified as *functionally-critical* ($\mathsf{funct\_crit}(link)$). Thus, the corresponding rule can be defined as

$$\mathsf{block\_link}(link_{i,j}) := \neg\mathsf{funct\_crit}(link_{i,j})(i \neq j) \quad (1)$$

If a link is not classified as *functionally-critical* but as *redundant*, it might be blocked, depending on the severity of the incident.

**Isolating a Host.** A host should not be isolated, if it is marked as *functionally-critical* or if isolation would affect any link classified as *functionally-critical*. This rule can be defined as

$$\mathsf{iso\_host}(host_i) := \neg[\mathsf{funct\_crit}(host_i) \vee \exists link_{k,l} \in L : \mathsf{funct\_crit}(link_{k,l}) \wedge (k = i \vee l = i)](i \neq j) \quad (2)$$

If a link is not classified as *functionally-critical* but as *redundant*, it might be still reasonable to isolate the host, depending on the severity of the incident. Furthermore, as mentioned in section 2.1, isolation of a host with the exception of specific links is also possible.

**Isolating a Switch.** A switch can only be isolated when each link classified as *functionally-critical*, *redundant* and *time-critical*, directed through this switch, can be redirected without using the switch. Moreover, for links marked as *time-critical*, the respective time requirements have to be met for their new paths and links classified as *redundant* have to be

ensured to stay redundant. This rule can be defined as

$$
\begin{aligned}
&\mathsf{iso\_switch}(switch) := \\
&\quad \forall link_{i,j} \in L : switch \in path_{link_{i,j}} \Rightarrow \\
&\quad [\mathsf{funct\_crit}(link_{i,j}) \Rightarrow \exists path_{i,j} : switch \notin path_{i,j}] \\
&\quad \wedge [\mathsf{time\_crit}_{\tau}(link_{i,j}) \Rightarrow \exists path_{i,j} : switch \notin path_{i,j} \\
&\qquad \wedge \mathsf{meets}_{\tau}(path_{i,j})] \\
&\quad \wedge [\mathsf{redundant}(link_{i,j}) \Rightarrow \exists path_{i,j}, path'_{i,j} : \\
&\qquad switch \notin path_{i,j} \cup path'_{i,j} \\
&\qquad \wedge path'_{i,j} \cap path_{i,j} = \emptyset](i \neq j) \qquad (3)
\end{aligned}
$$

For the sake of complexity reduction the above expression is reduced to its essential statements. For example, a link marked as *time-critical* and *redundant* must have an alternative path which meets the time requirements and is disjoint to the current backup path (in order to be redundant). The expressed rule does not yet take such combinations into account.

**Monitoring a Host.** The main issue in replicating data from hosts for monitoring purposes is the introduction of new traffic between the replicating switch and the monitoring system. Thus, the SDN management might only need to verify that no link classified as *time-critical* looses its real-time assurance when the path between the replication switch and the monitoring system is deployed.

**Monitoring a Switch.** When monitoring a switch, as many links directed over that switch should be monitored as possible. Assuming the switch can no longer be trusted, a prerequisite to monitor such a link is that its current path contains other switches which can act as replicating switches. Just like for "Monitoring a Host", the only time-sensitivity and availability restrictions for this action apply to the path from the replicating switch to the monitoring instance. Thus, for the replication path, the same time-sensitivity checks have to be conducted as explained for "Monitoring a Host".

**Virtualization.** Since different possibilities leveraging virtualization for incident response exist, which all need further research, rules for such techniques are out of scope for this paper.

## 2.4 Architecture

Earlier, we described what incident response actions can be executed with SDN (cf. section 2.1) and how they can be restricted. Extending the SDN4S approach, we developed an architecture to build our

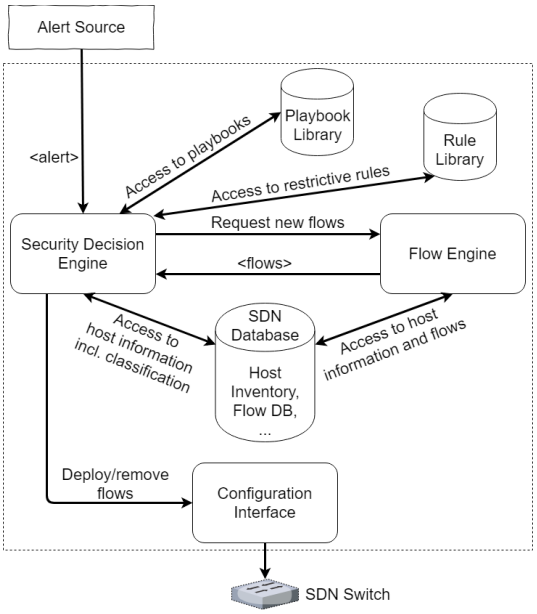


Figure 1: Overview of the here described SDN-IR architecture.

SDN-IR approach with common SDN platforms (cf. figure 1). We introduced the components *Security Decision Engine (SDE)* and *Rule Library* to the SDN4S approach. The SDE receives alerts, e.g. from an IDS. Upon receiving an alert, the SDE derives the matching playbook from the Playbook Library. Since the here described SDE relies on deterministic decisions, we assume the existence of a matching playbook for every possible received alert. Given the response actions suggested by the retrieved playbook and the type of assets (i.e. host, switch or link) affected by or responsible for the incident, the SDE obtains the respective restrictive rules (cf. section 2.3) from the Rule Library. Interpreting the rules, the SDE knows which checks to perform for what assets, in order to decide whether to perform an action or not. Like our basic rules defined in section 2.3, most rules will instruct the SDE to perform certain checks on assets with specific classifications. For this, the classifications of such assets have to be retrieved from corresponding inventories, like the Host Inventory. If the SDE decides to perform the response actions based on the check results, it generates flow entries either by itself, or by using the SDN's flow engine (cf. L2Switch of OpenDaylight, section 3), which depends on the specific implementation. Thus, the SDE is able to use the intelligence of the flow engine to find the best flows for the current topology, e.g. when links have to be redirected, or apply straight-forward flow entries (e.g. when a certain link has to be blocked). The flow entries are then deployed on or removed from the switches via the Configuration Interface (e.g.

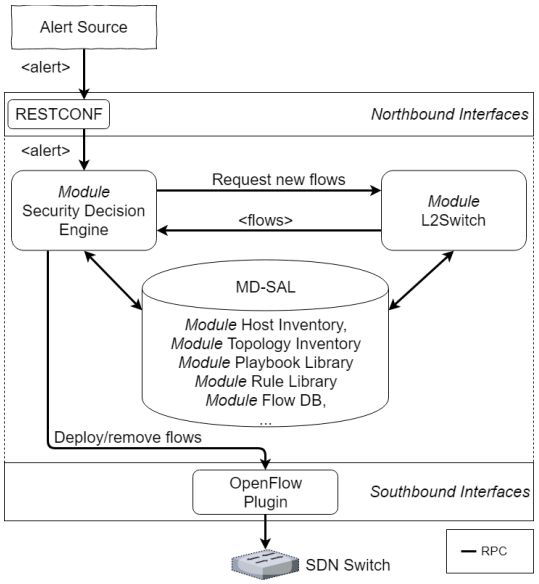


Figure 2: Simplified overview of the prototype implementation of the here proposed SDN-IR architecture (cf. section 2.4).

NETCONF[1], OpenFlow (McKeown et al., 2008) or SNMP[2]). The incident response flow entries are defined with a higher priority than common flow entries. Therefore, old flow entries, colliding with the SDE's flow entries are ignored when packets have to be forwarded. This way, the old state of the network can be recovered easily if the measure gets repealed. The old flow entries will be immediately active again after the SDE's flow entries were removed. For most actions, this strategy might be preferable to removing original flow entries. Furthermore, when using NFV an implementation of the SDE could leverage existing security controls like firewalls as well. For actions which can be performed by such controls, this is a more consistent approach than directly applying flow entries.

## 3 EVALUATION

Our goal was the identification of possible pitfalls when it comes to the implementation of our concept on a common SDN platform. Thus, for evaluating our concept, we applied and extended the well established *OpenDaylight* (*ODL*) framework (Medved et al., 2014). The modular architecture of ODL allowed us to build a customized SDN management platform. A simplified overview of our prototype implementation can be found in figure 2. To conserve space, we only give a brief insight into our implementation.

In our implementation, the alerts are passed to ODL via POST REST calls which consist of an asset id (e.g. *openflow:1* cf. figure 3), an alert category and a priority value. On the target side of this REST call a RESTCONF (Bierman et al., 2017) interface receives the alert. The RESTCONF interface is applied as a so called northbound interface in ODL which is responsible for the communication between ODL and external applications or users.

ODL's database management is called Model-driven Service Abstraction Layer (MD-SAL)[3] which uses the modelling language YANG (Bjorklund, 2010) to define the database scheme. Among other things, it is used to store flows, switch configuration and topology information. We added our own models to store additional host information (like identifiers, status flags and classifications (cf. section 2)), submitted alerts, the playbook and rule libraries. Furthermore, we extend the Flow Database (cf. figure 2) to store classification for links. This is a practical strategy, chosen under the assumption that all configured links are represented by persisted end-to-end flow entries. In consequence, for every link which should be classified, e.g. as *functionally-critical* or *redundant*, our prototype demands flow entries being generated in order to add the classifications to them.

The received alert first gets persisted to the database. Afterwards, the SDE gets invoked and is given the alert's identifier. The SDE retrieves the new alert from the database and queries the Playbook Library for actions to perform for the given combination of the alert category, asset type and priority. The priority is used as an indicator for the severity of the incident. In our case, the type of asset, can be determined via interpretation of the asset identifier given as argument of the REST call. The playbook received as the response can be seen in listing 1.

```
SWITCH
    DENIALOFSERVICE
        Priority.HIGH
            ActionType.ISOLATE
            ActionType.NOTIFY
        Priority.LOW
            ActionType.MONITOR
            ActionType.NOTIFY
```

Listing 1: Example playbook entry for switches on *DENIALOFSERVICE* suspicion, which defines response actions for two different alert priorities.

Subsequently, the SDE retrieves the rules from the Rule Library using the combination of asset type and

[1]https://tools.ietf.org/html/rfc6241.html
[2]https://tools.ietf.org/html/rfc1157

[3]https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Developer_Guide

action to perform (listing 2 shows an example rule for asset type `SWITCH` and action `ISOLATE`).

```
SWITCH
    ISOLATE
        LINKS
            functionally-critical
                {redirectable}
            redundant
                {redirectable}
```

Listing 2: Example rule entry for switch isolation.

The resulting requirements are then checked by ODL's built-in functionality, like the *NetworkGraphService*. For example, in listing 2: "links classified as *functionally-critical* or *redundant* have to be *redirectable*". If the action's requirements can be mapped, the SDE generates the respective flow entries, e.g. to isolate the switch, and requests additional flows from the L2Switch, e.g. by creating a temporary topology and letting the L2Switch generate the new flows for this topology. The flow entries are sent to an OpenFlow plugin (McKeown et al., 2008), which we chose as the SDN controller's interface towards SDN switches (southbound interface). This interface matches the Configuration Interface of figure 1.

Despite the evaluation regarding the feasibility of the implementation of our concept with current SDN platforms, we were interested in its impact to the network's performance. To be able to set up various topologies, we used Mininet[4] in version 2.2.0 for a network emulator, which supports SDN switches and external SDN controllers. We configured Mininet using our ODL implementation as external controller, communicating with it via OpenFlow 1.3, using a bandwidth limit of 10 MBit/s and a delay of 10 ms. Afterwards, we conducted several tests, namely isolating hosts and switches, redirecting traffic of hosts and switches to the monitoring system as well as blocking various links. These actions were performed with and without classifications and on various topologies. Furthermore, the latency and bandwidth during the execution of the response actions were measured. One of these topologies can be found in figure 3. Amongst others, we used this topology to test the isolation of a switch. This test had the most significant, measurable impact on the network, which is the reason why here we use it as an example of our results.

Figure 4 depicts the measured latency (using the *ping* command) between *host1* and *host2* when the isolation of switch *openflow:1* is performed in second 12.

Before the isolation, the two hosts communicate over the path (*openflow:2*, *openflow:1*, *openflow:3*).

[4]http://mininet.org

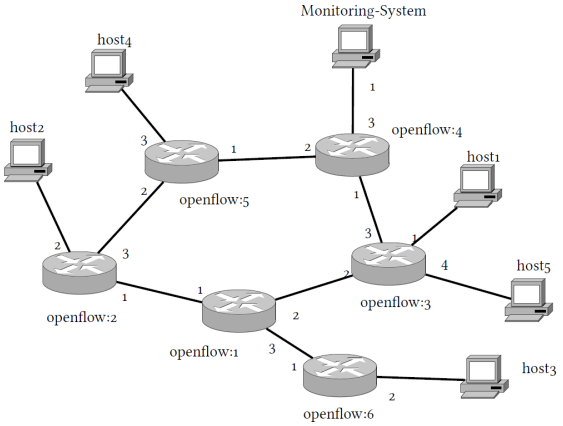


Figure 3: Example topology we used for evaluation of our prototype. The numbers next to the SDN switches index the physical ports of each switch.

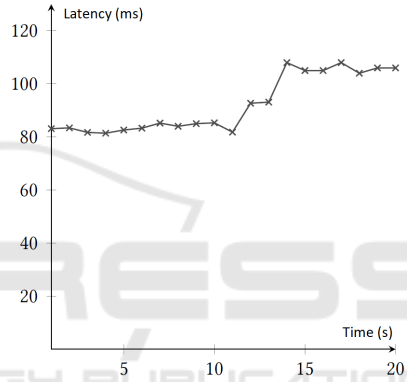


Figure 4: Latency measurement when isolating switch *openflow:1* in second 12.

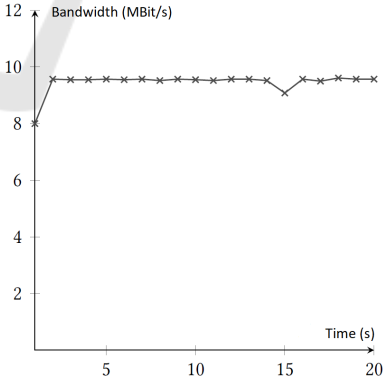


Figure 5: Bandwidth measurement when isolating switch *openflow:1* in second 12.

After the isolation the new path (*openflow:2*, *openflow:5*, *openflow:4*, *openflow:3*) was used. The picture shows that after the switch is isolated in second 12, the latency slightly increases due to the extra hop in the new path. In contrast, the bandwidth (measured using *iPerf*) remains unchanged, besides a

small slump in second 15 which can be traced back to the adaptation to the new path (cf. figure 5).

During the validation we identified open tasks regarding the implementation in terms of a productive usage. Our prototype currently only supports the object-oriented definition of rules. This is closely aligned to the way our program conducts the feasibility analysis. In the future, a more general approach would improve implementations, e.g. by providing a respective specified rule language. Furthermore, if a playbook's action is refused, it is currently not possible to configure alternative actions. Even though it is possible to add additional playbooks for the same incident-asset-classification combination, they currently cannot be prioritized. In addition, whether an alternative path can meet the time criteria of a real-time link has to be determined by the controller before the alternative path can be selected and suggested to the SDE. This might then result in an SDE-L2Switch negotiation to find alternative paths. However, currently our implementation is lacking the support of the real-time calculation (and thus the negotiation). As part of the project FlexSi-Pro[5] we are currently developing a solution for this issue using Time-Sensitive Networking[6]. In section 3, we proposed the strategy to mimic the classification of links by classifying their respective flow entries. In production, it would be more reasonable to have a dedicated link representation within the SDN database, e.g. within the topology inventory, and adding classifications there. This slightly increases the necessary development effort and the complexity of determining the links, but separates the link configuration from the currently available flow entries and therefore does not require end-to-end flow entries being preconfigured for every classified link. Thus, we are planning to detach the link representations from flow entries.

We are currently addressing these open tasks in the mentioned project FlexSi-Pro and will deploy and further develop our prototype in our ICS test laboratory (Pfrang et al., 2016).

## 4 CONCLUSIONS

In this work, we described an SDN-based solution for automated incident response in flexible ICS networks. In contrast to previous research, our proposed solution takes into account the multiple restrictions for automated incident response in an ICS, utilising restrictive rules and asset classification. A basic, adaptable set of such rules is described in this paper. Instead of reinventing the wheel, we designed our solution to be compliant with the already existing SDN4S concept and built a prototype demonstrating the feasibility of the implementation of our approach on common SDN platforms. Based on this prototype, we were able to evaluate our concept to identify remaining issues and potential starting points for future research. With this paper, we presented an enabler for SDN-based incident response in environments which have special restrictions, regarding incident response actions affecting host, network components and communication links.

[5]https://www.wibu.com/uk/flexsi-pro.html

[6]http://www.ieee802.org/1/pages/tsn.html

## REFERENCES


Bierman, A., Bjorklund, M., and Watsen, K. (2017). RESTCONF Protocol. RFC 8040, RFC Editor. Last accessed on Dec, 2018.

Bjorklund, M. (2010). YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, RFC Editor. Last accessed on Dec, 2018.

Bromiley, M. (2017). The Show Must Go On! The 2017 SANS Incident Response Survey. Analyst paper, SANS. Last accessed on Dec, 2018.

Cichonski, P., Millar, T., Grance, T., and Scarfone, K. (2012). *Computer Security Incident Handling Guide : Recommendations of the National Institute of Standards and Technology*. National Institute of Standards and Technology.

Di Lallo, R., Griscioli, F., Lospoto, G., Mostafaei, H., Pizzonia, M., and Rimondini, M. (2017). Leveraging SDN to Monitor Critical Infrastructure Networks in a Smarter Way. In *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network Management*, pages 608–611, Piscataway, NJ. IEEE.

Kim, H. and Feamster, N. (2013). Improving Network Management with Software Defined Networking. *IEEE Communications Magazine*, 51(2):114–119.

Koulouris, T., Casassa Mont, M. and Arnell, S. (2017). SDN4S: Software Defined Networking for Security. Report, Hewlett Packard Enterprise. Last accessed on Dec, 2018.

Martins, J. S. B. and Campos, M. B. (2016). A Security Architecture Proposal for Detection and Response to Threats in SDN Networks. In *Proceedings of the 2016 IEEE ANDESCON*, pages 1–4, Piscataway, NJ. IEEE.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69.

Medved, J., Varga, R., Tkacik, A., and Gray, K. (6/19/2014 - 6/19/2014). OpenDaylight: Towards a Model-Driven SDN Controller architecture. In *Proceeding of IEEE*

*International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6. IEEE.

Pfrang, S., Kippe, J., Meier, D., and Haas, C. (2016). Design and Architecture of an Industrial IT Security Lab. In *Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 114–123. Springer.

Piedrahita, A. F. M., Gaur, V., Giraldo, J., Cardenas, A. A., and Rueda, S. J. (2018). Virtual incident response functions in control systems. *Computer Networks*, 135:147–159.