Network Trace Generation for Flow-Based IDS Evaluation in Control and Automation Systems

Gorby Kabasele Ndonda[a,1,*], Ramin Sadre[a,2]

[a] *UCLouvain, INGI Dpt., Réaumur Place Sainte Barbe 2, 1348, Louvain-la-Neuve, Belgium*

Abstract

The increasing number of attacks against Industrial Control Systems (ICS) have demonstrated that there is a need to secure such systems. Unfortunately, directly applying existing ICT security mechanisms is hard due to constraints of ICS, such as availability requirements or resource limitations of the field devices. Thus, the solution preferred by researchers is the use of network-based intrusion detection systems (N-IDS). An issue that many researchers encounter is how to validate and evaluate their N-IDS since it is very difficult to get access to real and large ICS for experimentation. The few public traffic datasets that could be used for off-line experiments are either synthetic, collected at small testbeds or not suited for network experimentations.

In this paper, we present a tool to generate network traces based on statistical properties that the tool extracts from empirical traces. We demonstrate its usability by applying it to an empirical trace collected at the Heating, Ventilation and Air Conditioning (HVAC) management system of a university campus and using the generated traces to evaluate several IDS published in the literature. We make the original trace available to other researchers. To our knowledge, we are the first to publish a network dataset collected at a real and operational control and automation system.

Keywords: Network Security, Industrial Control System, Intrusion Detection System

1. Introduction

Industrial Control and automation Systems (ICS), such as SCADA, have constraints different from traditional enterprise IT systems, especially with regard to availability. Often, end hosts have to operate continuously for several decades and it is not uncommon to see outdated hardware and software, as updating is difficult and expensive. For this reason, Network-based Intrusion

*Corresponding author
[1] gorby.kabasele@uclouvain.be
[2] ramin.sadre@uclouvain.be

Detection Systems (N-IDS) have been the preferred solution by researchers to secure ICS [1, 2] since, in contrast to host-based IDS, they do not require modifications of the end hosts.

Among techniques for N-IDS, *flow-based* methods have been employed for several years to build IDS for the Internet [3] and also specifically for ICS [4, 5, 6]. Flow-based IDS see network traffic as a set of flows, where a flow is defined as a sequence of packets with common characteristics. For example, a flow can be defined as all packets with identical source and destination address. Only aggregated metrics of the flows (such as the total byte size) are fed to the IDS. Information about individual packets, especially their payload, is not collected. Flow-based IDS are attractive for ICS because of the abundant proprietary protocols used by the latter, meaning that the same IDS can be used in any ICS network. Moreover, with the rise of security mechanisms for ICS using encrypted communication in constrained environments [7, 8], other detection mechanisms like application payload inspection will become more and more difficult to perform while flow based detection will be still relevant.

One of the problems that researchers are facing is how to evaluate and validate their IDS algorithms. Getting access to a real and operational ICS is difficult. Instead, most researchers rely either on experiments in testbeds and simulators or use traffic traces collected in such environments for off-line experiments [9, 10, 11]. The advantages of those approaches are the reproducibility of experiments, the flexibility to test different configurations, and the fact that no real system is put in danger. However, they require that the used environments accurately reflect the behavior of a real system, which is challenging to ensure. Publications relying on empirical data from *real* systems are much rarer and those few that exist have not published their datasets [4, 6].

The contribution of this paper is two fold. First, we describe a dataset that we have collected at the Building Management System (BMS) of a university campus and that we have made publicly available, so it can be also used by other researchers. To our knowledge, while some telemetry captures exist [12], our dataset is the first public network dataset from a real, operative automation system. The dataset contains the headers of the network packets exchanged between field devices, control servers and human machine interfaces (HMI) of the campus' Heating, Ventilation, and Air Conditioning system (HVAC). The dataset does not contain packet payloads due to privacy and security reasons, but it can be used for purposes where traffic traces on packet header level or flow level are sufficient, such as flow-level traffic characterization. To the best of our knowledge, the dataset only contains benign traffic. This means that it cannot be directly used to validate intrusion detection systems. The description of the dataset that we provide in this paper is based on a report accompanying the dataset and containing further details on the technical aspects of the data collection and preparation process [13].

Second, we present an approach to generate new network traces containing legitimate traffic as well as malicious traffic. Being able to produce new traces allows to evaluate how IDS would perform in different situations or in the presence of unexpected side-effects. Our approach takes as input a traffic

trace from which it extracts the parameters for the generation process of the legitimate traffic. For the malicious traffic, attacks are chosen from a pool of attacks known in the literature. We show the usefulness of our approach by applying it to different datasets and using the generated traces to evaluate several flow-based IDS described in the literature.

The structure of this paper is as follows. Section 2 gives our motivation for this work. Related work is discussed in Section 3. In Section 4, we describe the public empirical dataset in order to clarify and motivate the design choices taken in the trace generation process depicted in Section 5. The experimental evaluation is provided in Section 6. Finally, the paper concludes in Section 7.

2. Motivation

An ICS controls and monitors industrial processes. It is generally composed of a central server connected via a LAN or WAN to field devices which in turn are connected to sensors and actuators through a field bus. Large parts of the control logic tend to be implemented on the field devices and the role of the server is to periodically retrieve the sensor data from them and to set general process parameters.

Although modern ICS use the same TCP/IP-based network technologies as traditional Information and Communications Technology (ICT) networks, important differences exist [14]. Most of the communication in an ICS is caused by automated processes, meaning that ICS networks are much more stable in terms of network topology and traffic exchanges than a ICT network where human activities are more dominant. Additionally, there is a difference in security management. In an enterprise network, the focus is on protecting the data or processes on the servers. Employee workstations must be protected, too, but they are often just seen as entrance points for attacks. In ICS, all hosts are of equal importance: Attacking a server can disturb the high-level process, but an attack against a field device impacts the machines connected to it.

Because of these differences, existing IDS solutions from ICT environments have to be carefully tested against ICS datasets to see if there are still valid in that context. Likewise, new IDS designs for ICS networks have to be evaluated. Many public network datasets exists for ICT networks [15], but to our knowledge nothing comparable exists so far for ICS, which is a major problem for researchers working on ICS security. This highlights the need to generate ICS datasets. Such tools exist for ICT networks. However, we argue in the discussion of related work (Section 3) and in the presentation of the results (Section 6) that our dedicated approach is more suitable to ICS than tools designed for the generation of Internet-like traffic.

We decide to focus on datasets for *flow-based* IDS because of their suitability and attractiveness for ICS. Indeed, compared to other IDS solutions, e.g. Deep Packet Inspection (DPI), they are independent from the details of the concrete application protocol. A DPI-based IDS inspects packet payloads to detect attacks, meaning that a dissector must be implemented for every protocol. This is a major drawback as it is time-consuming and often not feasible due to the

heavy use of proprietary protocol in ICS. Moreover, even though current ICS are weakly secured in general, we see a trend toward the usage of secured protocols, especially with the upcoming Industry 4.0, therefore we expect that the interest in flow-based IDS will increase as payload inspection will be more difficult to do because of encryption.

We identify three requirements to a trace generator for the evaluation of flow-based IDS:

1. **Accuracy**: The generated traces must replicate the behavior of real ICS at flow-level. In other word, the flow traffic patterns in the generated traces must be similar to the ones one could expect from a real ICS. As we will explain in Section 5.1, our trace generator replicates bidirectional flow-level characteristics as well as certain packet-level characteristics. Although the latter are not strictly needed by pure flow-based IDS, their correct emulation increases the flexibility of our trace generator because it does not impose a unique flow definition on its users. However, we do not require a perfect replication of packet header fields and signaling packets, such as ACK packets.
2. **Diversity**: The generated traces must contain benign and malicious traffic. The synthetic traces are generated in order to evaluate flow-based IDS, in other words, their purpose is to assess the ability of a flow-based IDS to make the distinction between legitimate and malicious traffic. Therefore, these two kinds of traffic must be included in the generated traces. In real-world scenarios, targets of attack traffic will reply to such traffic, so we want that the malicious traffic included in the trace come from both the attacker and the victim.
3. **Non-determinism**: Multiple runs of the generator must not lead to the same trace, thus representing the non-deterministic nature of a network. By mostly consisting of machine-to-machine communication, the traffic in ICS is fairly periodic and stable, however the quirks of the networks (failures, delay, etc) are sources of non-determinism. For two generated traces, we want that the characteristics of individual packets are not identical, while still respecting the overall per-flow empirical distributions observed in the original trace.

3. Related Work

Traffic generation is an important tool for analyzing the performance of networked systems [16, 17, 18, 19]. Typically, the generated traffic is used as workload, for example to identify the maximum throughput of a particular network design. Many proposed solutions, such as [16], focus on uni-directional traffic generation in order to simplify the generation process, assuming that the traffic from a server to its clients dominates the perceived performance of the studied system and that therefore the other direction can be neglected. Such an approach is not realistic to evaluate IDS. In an ICS, both directions of the communication between a control server and a field device are equally

important, therefore, in the evaluation of the IDS, both endhosts must behave like they would in a real environment. In [20], the authors implement two-way communication by accurately replaying existing traces in order to assess the quality of a network in an objective way. In contrast, our goal is to deliberately generate traces that differ from each other in a certain way, so they can be used to evaluate IDS under different conditions. There exist several open-source tools that can be used to generate bi-directional traffic. Like [20], Tcpreplay [21] replays network traffic previously captured and does not fulfill our need for non-determinism (Section 2). Iperf [22] is a speed test tool. While it could be used to replicate communication between two end hosts at flow-level, there is no way to directly control the Packet Size distribution (PS) and the Inter Packet Time (IPT) distribution, which is a desirable feature as we will discuss in Section 5. D-ITG [23] allows to choose the PS/IPT distributions from a predefined set of distributions (Normal, Gamma, etc). In contrast, our approach is able to replicate arbitrary empirical distributions.

More related to traffic generation for IDS evaluation are [24, 25]. In [24], the authors model SSH brute-force attacks at flow level with a Hidden Markov Model. However, they focus only on malicious traffic. In [25], the authors present a framework to create attack profiles defined by exploits and the propagation method. When generating traffic, packet header and payload are built from blocks and a validator monitors the target responses to ensure their correctness. The work is extended in [26] to integrate legitimate traffic into the generation but the communication is one-directional. Finally, there exist a few works on the generation of network traffic for the evaluation of IDS for ICS [27, 28, 10]. All those works can be classified as application-level generators, as they generate traffic such that the packet payload is correct according to the application protocol specification. The protocol in question is Modbus [29], a widely used SCADA protocol. In [27], the authors parse the list of detection rules of Snort to automatically generate Modbus packets with malicious payload that matches those rules. The authors of [28] and [10] generate synthetic Modbus traces with attacks mixed in. Although these approaches generate packets with correct application-level payload, they have to rely on virtual environments and testbeds to simulate the overall behavior of an ICS and, therefore, suffer from the problems described in Section 1.

4. A public empirical network trace

To evaluate our trace generation approach with real data, we use a dataset that we have collected at the HVAC system of a university campus. We have published this dataset, so it can be used by other researchers. To our knowledge, this is the first publicly available network dataset from a real, operative control and automation system. A detailed description of the dataset and the methods used to collect it is given in a separate report [13]. In the following, we will repeat parts of it because they contribute to the understanding of some of the design decisions taken in Section 5.

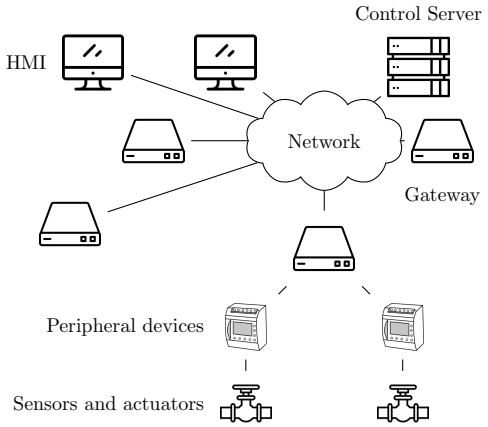


Figure 1: BMS network overview

The trace contains the network communication of a HVAC system. The system is fully automated, with a server communicating with peripheral devices deployed on the campus to control heating in rooms of the university. The BMS, topology-wise, follows a typical design for automation systems [30]. A communication network interconnects the control server, the Human Machine Interface (HMI) stations, and the peripheral devices. Sensors and actuators are attached to the latter making them similar to Progammable Logic Controllers (PLCs) in SCADA systems: They collect and send sensor data to the server and their behavior can be (manually) influenced from the HMIs, but most of the time they execute control programs with setpoints defined by the server.

The network is logically isolated from the rest of the campus network. The protocols are proprietary and use TCP/IP. Peripheral devices are not directly connected to the network. Instead, a group of devices is connected to a gateway which acts as an end point for the TCP connection from the control server (Figure 1). This allows to also use fields devices which are not IP-capable. There are eight such gateways in the dataset.

Details on the traffic collection and data preparation process can be found in [13].

4.1. Traffic characteristics

This section presents the main characteristics of the trace. We will discuss some of those characteristics in terms of unidirectional flows. We define a flow using the usual five-tuple (source/destination IP address, source/destination port, transport protocol). When we refer to flow size or packet size in the following, we always mean the size of the payload, excluding headers. For TCP flows, we only consider the ones where at least one byte of payload was exchanged, thus removing connections which failed to be established. Note that our flow definition ignores TCP flags (SYN, FIN etc.), i.e. we follow more or less

Host Type	Nbr	Communicates with	Payload	Nbr pkts
HMI stations	7	Control Server	6.7GB	24.7M
Control Server	1	Gateways	387.7MB	11M
Gateways	8	Periph. devices	–	–
Periph. devices	58	–	–	–

Table 1: HVAC summary

the definition of a flow record in Netflow/IPFIX [31] with active and inactive timeout set to infinity.

4.1.1. Overview

The dataset covers a period of 7 days thus capturing the behavior of the BMS on weekdays as well as during the weekend. Table 1 shows a summary of the HVAC system. The right three columns give basic statistics of the bidirectional communication between HMI↔server and server↔gateways, respectively. The peripheral devices are not IP-capable — there is no traffic between the peripheral devices and the gateways in the dataset. In total, the dataset contains 11.2 GB of packet data (corresponding to 4GB of packet headers) from 47 IP addresses and around 23,000 flows. As expected from an automation network, flows are relatively small; the largest one transported 614.7 MBytes (5M packets) over one week. The average flow size is 342 kBytes (1725 packets). Figure 2 shows the empirical CDF of (a) the number of packets per flows, (b) the number of bytes per flow, and (c) the flow duration. There is a significant number of flows with a very long duration and a large number of packets, although the distribution of the number of bytes per flow is rather conservative. This is typical for automation systems where the server and the peripheral devices periodically exchange small control packets through more or less permanent TCP connections.

4.1.2. Application protocols and flow stability

Most of the observed traffic stems from communication between the HMI stations and the control server. The HMI stations and the server communicate through several ports. First, they exchange RPC for Distributed Computing Environment (DCE/RPC) packets [32]. The control server acts as the DCE/RPC server on port 135. In addition, the HMI workstations communicate with the control server on port 50000. The protocol used for the communication between the server and the gateways is proprietary (port 2499).

As discussed in Section 2, ICS networks are generally more stable behavior-wise. This is also true for the BMS network studied by us. We show in Figure 2d the number of flows seen per hour. For this figure, we have counted flows directly related to the HVAC system, i.e. flows between the control server, the HMI workstations and the gateways, separately from other UDP and TCP flows. Except for the first hour when we start the capture, the number of flows discovered for the HVAC system is mostly constant. Actually, flows from the

gateway to the control server lasted for the whole duration of the capture and no new flow were created. The peaks that can be seen on the HVAC result from the change of the port used by the control server for DCE/RPC. The number of TCP flows discovered usually does not exceed 500 except at two points where we can see high peaks. The non-HVAC TCP and UDP flows are mostly HTTP/HTTPS and DNS flows created when the HMIs check for software updates. The peaks are caused by successful updates.

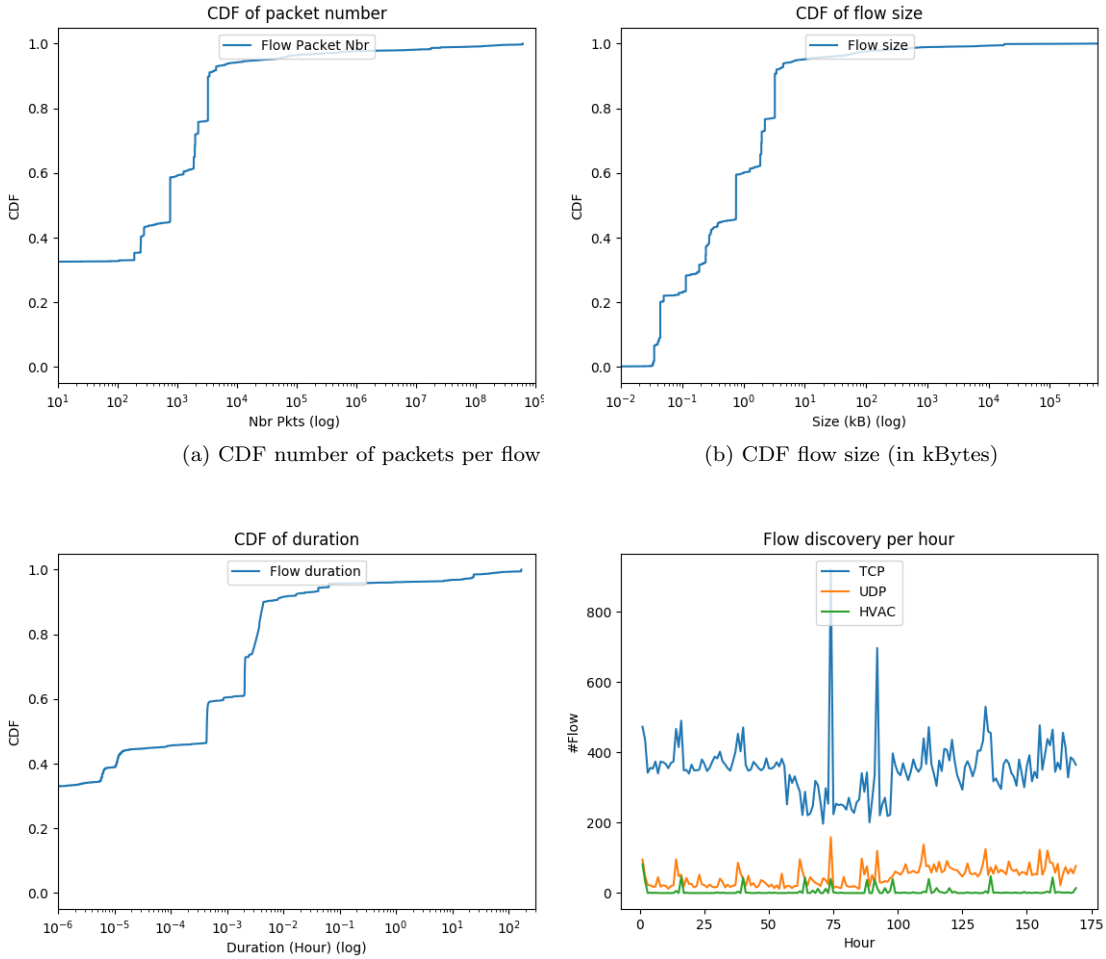
4.1.3. Network activity time series

As mentioned before, the peripheral devices are connected to the IP network through gateways. The devices differ in age and capabilities and therefore depict different communication characteristics. This is illustrated in the following with two gateways X and Y which connect 11, resp. 13, peripheral devices. Figures 3a and 3b show timeseries of the number of packets and bytes exchanged per hour between the server and the two gateways. We note that, despite having a comparable number of devices attached to them, the number of packets observed at gateway Y is considerably higher than at X. A similar observation can be made for the number of exchanged bytes. We also notice how differently the packets are generated in both directions. In gateway X, the number of packets sent to the server is close to the number of packet sent in the other direction. In fact, most of the packets sent by the server to X are ACK segments. We can see that the server behaves differently in the case of gateway Y: It actively sends segments with control commands to the gateway. Since the gateways are connected to devices of different models, it means that the devices connected at gateway Y are more data intensive.

Finally, we observe diurnal and weekly patterns in the timeseries. The measurement was started on a Friday at around 2PM, therefore the first 55 hours of low activity correspond to the weekend. This behavior is understandable when considering the nature of the physical processes controlled by the BMS, but it should be noted that not all automation systems behave like this [33]. Interestingly, gateway Y shows lower activity during the night than during the weekend although the buildings are not used in both situations. Our assumption is that there is a sleep mode enabled during the night which decreases the frequency at which data are exchanged.

4.1.4. Packet size distribution

SCADA network protocols such as Modbus/TCP [29] require that the payload fits in a single IP packet to avoid fragmentation. These protocols aim at low latency or even real-time usage, so packets tend to be small. We study the packet sizes of the packets sent from both gateways to the server. Gateway X sent 255,251 packets and gateway Y sent 5,183,021 packets. Figure 3c shows the normalized histograms of the observed packet payload sizes in bytes for each gateway. Typical for control systems, the packets are rather small in both cases (maximum 170 bytes) and only a few different sizes can be observed. Again, there is a difference between the two gateways. Most of the packet sent by X



(a) CDF number of packets per flow

(b) CDF flow size (in kBytes)

(c) CDF flow duration (in hours)

(d) Flows seen per hour

Figure 2: Flow characteristics

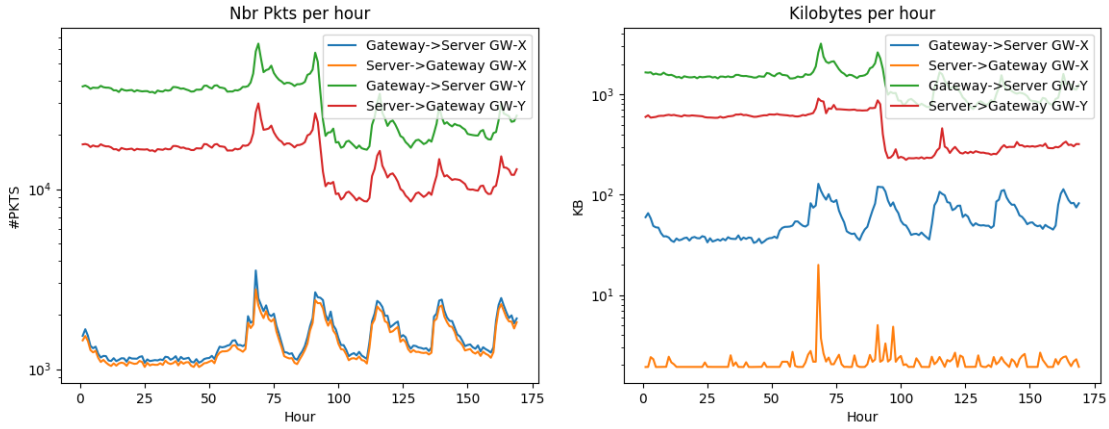
contain actual data and only a small percentage (approx. 20%) of empty ACK, while Y sends more than 2 million empty packets.

5. The traffic trace generator

5.1. Overview and design choices

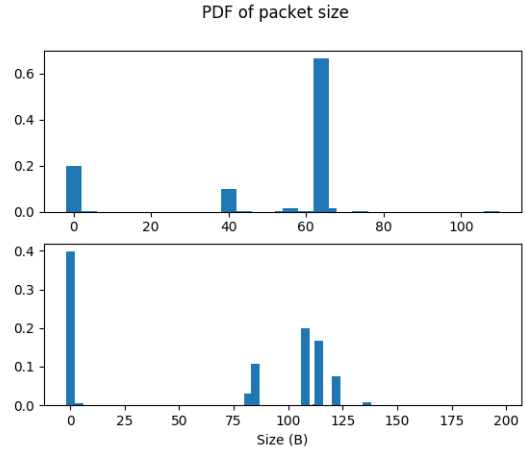
As mentioned in Section 3, there exist many different traffic generator. They operate at different levels [19]:

- *Application-level* generators emulate communication patterns of network applications. A detailed knowledge of the applications/protocols is required to create such generator. In ICS such knowledge is available for protocol like Modbus but many other are proprietary and undocumented.



(a) Packets per hour

(b) kBytes per hour



(c) Packet size distribution - Top: Gateway X, Bottom: Gateway Y

Figure 3: Traffic observed at gateway X and Y

- *Flow-level* generators replicate traffic characteristics on flow level, such as the number of packets and the number of bytes transferred per flow. Their advantage is that no knowledge of the concrete application protocols is needed.
- *Packet-level* generators replicate characteristics of individual packets, such as packet size (PS) and inter-packet time (IPT). Therefore, packet-level generators can be also used as flow-level generators but not vice versa.

Generators can also be also differentiated by their generation process. It can either be model-driven, meaning that the generation is done according to a model built a priori, or trace-driven, i.e. the generated traffic is a replay of a previously recorded trace [20].

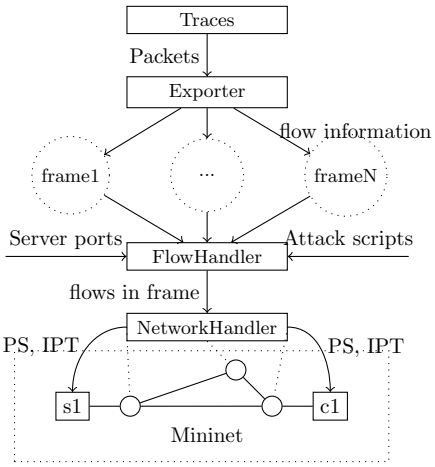


Figure 4: Workflow of the trace generator

As explained in Section 1, our goal is to generate traces for the evaluation of flow-based IDS. This would suggest that a flow-level generator is sufficient. However, there is no unique definition of a flow used by all researchers. Although many define a flow with the IP header five-tuple (source/destination addresses, source/destination port, protocol number), the IPFIX specification [31] is much more general. The traffic generator that we present in the following operates on packet level for maximum flexibility. Its input is a packet trace from which it creates a model of the packet characteristics for each connection found in the input trace. These models are then used to generate a new traffic trace.

At this point, we have to ask how closely the generated traces should resemble the original input trace. An ICS network can have many quirks that are unrelated to the actual control and automation processes, such as the set of TCP options used by the hosts or the way acknowledgment packets are generated. We do not want to create exact copies of the input trace; our goal is to create traces that can be considered as valid ICS traces, but are sufficiently different from the input trace so they can be used for IDS evaluation. For this reason, the models used in our trace generator only capture the PS and IPT distributions of those packets of the input traces that actually contain application data, meaning that signaling packets such as empty ACK packets are not taken into account. Of course, a correct traffic trace must also contain signaling packets; we emulate a virtual network in Mininet [34] to generate them.

Figure 4 gives the general workflow of our traffic generator. We explain the different steps of the generation process in the following.

5.2. Exporter

The *Exporter* reads an input packet trace and exports flow information that will be used by the next stage, the FlowHandler. The Exporter first splits the input packet trace into blocks of identical duration, called *frames* in the

following. This is necessary because the characteristics of the packets may vary considerably from frame to frame: As shown in Section 4, the behavior of a control and automation network system can depend strongly on the time of the day. For each frame, the Exporter performs the following steps:

1. Packets are grouped according to the traditional 5-tuple flow definition mentioned above, i.e., packets with identical source and destination address, source and destination port, and protocol number, are part of the same flow.
2. For each flow, general flow characteristics (number of packets, duration, etc.) are calculated and the list of PS and IPT values of all packets of the flow are built.
3. The list of flows and the computed flow data are forwarded to the FlowHandler.

Note that the Exporter has to be executed only once for an input trace. The flow information can be re-used whenever a new trace with similar characteristics should be generated.

5.3. FlowHandler

The *FlowHandler* processes the flow information exported by the previous stage frame by frame. Its first action is to group the flows into bidirectional flows, i.e. a flow is matched with the corresponding flow in the opposite direction. Then it determines for each bidirectional flow which host was the initiator of the communication. Like traditional Internet protocols, many ICS protocols follow a client-server model with the client being the initiator of the connection. Identifying who is the client and who is the server is not always obvious in a traffic trace if the protocol is UDP based or if the TCP handshake has been missed at the start of the capture, which is likely to occur for long connections (see Section 4.1). To cope with that, the user can provide a list of known server ports. If the FlowHandler finds a flow with ports that are not in the list, it considers that the source of the first packet in the bidirectional flow is the client host. Whenever the FlowHandler has identified a new flow in the frame, it instructs the next stage, the NetworkHandler, to create packets similar to those in the flow. For each flow, it will forward the following information to the NetworkHandler:

- The five-tuple identifying the flow. The IP addresses of the input trace are replaced according to a list of IP addresses provided by the user.
- The start time of the flow. The generator tries to replicate the start times of the flows from the original trace. In practice, this is not necessarily done with perfect precision because the NetworkHandler needs time to configure the network emulation if a flow has not been seen before.
- The number of packets to generate for that flow in the current time frame. For each packet, the FlowHandler will sample the PS and IPT from the empirical PS and IPT distributions of the flow in the input trace.

- Whether the flow ends in the current frame. The FlowHandler peaks into the next frame to see whether the flow continues after the current frame. If not and the flow uses TCP, it will notify the NetworkHandler that the TCP connection should be closed after the last packet of the current frame.

To sample the PS, we estimate its probability mass function (pmf) from the histogram of the empirical data. Sampling new PS values from the pmf can be done efficiently since, as illustrated in Figure 3c and 3c, the communication in an ICS is often very regular with a few dominant packet sizes. We follow a different approach for the inter-packet times since time is continuous and even large input traces do not contain enough packets to obtain a useful probability density function (pdf) directly from the empirical inter-packet times: Instead, we use Kernel Density Estimation (KDE) to estimate the pdf.[3] KDE is a non parametric technique that estimates the pdf of a random variable by weighting the distances of all empirical data points for each possible value of the random variable [35].

5.4. NetworkHandler

As explained above, the *NetworkHandler* receives instructions from the FlowHandler on how to generate traffic in the current time frame. It is important to note that these instructions are not enough to directly output a traffic trace: Details on the packet header fields, e.g. TCP sequence numbers and options, and information on signaling packets, such as TCP SYN and ACK packets, are not provided by the FlowHandler. In order to generate a complete and consistent network trace, the NetworkHandler maintains a virtual network emulated with Mininet [34]. Mininet is a network emulator leveraging Linux network namespaces to create virtual networks.

This approach has several advantages. First, instead of simply replaying the input trace and mixing it with attack traffic in a separate step, we generate both the normal traffic and the attack traffic (see the next sections) in the same virtual network. This is important if a host exchanging normal traffic with other hosts is also sending attack traffic or responding to attack traffic. Our approach guarantees that in both cases, all traffic sent by a host has the same TCP/IP stack fingerprint. Otherwise, normal traffic and attack traffic (as well as the response traffic) could differ in unwanted ways and influence the detection result, since the TCP/IP stack can have an impact on the packets characteristics (e.g TCP Options) and therefore influence the flow properties. Second, the usage of a network emulator simplifies the configuration and management of the virtual network. Mininet enables to configure the IP addresses of the simulated hosts without having to adapt the routing table of the machine since the emulated network is isolated from the machine hosting the generator. Being able to manipulate the IP addresses allows to generated more realistic trace where

[3] We could also fit a parametric distribution, e.g. a Γ distribution, but there is no guarantee that a distribution fitting one ICS would work well for another ICS. We decided to use KDE since it does not require prior knowledge of the shape of the distribution.

communication occurs from hosts coming from different network, for example the control server contacting external servers for OS updates.

The NetworkHandler creates a virtual host in the emulated network for each IP address that appears in the instructions sent by the FlowHandler. This is done dynamically: When the NetworkHandler sees a new IP address, it runs a setup script that creates and configures a new host for that IP address. Once the host is ready, the NetworkHandler forwards it the traffic generation instructions that it has received from the FlowHandler for that source IP address. Since the hosts share a single filesystem and a single process ID space, the NetworkHandler can communicate with them through Inter Process Communication (IPC).

The topology of the emulated network is relatively simple: We use a variation of the dumbbell topology [18] with three interconnected switches named A, B, and C in the following. New hosts are either attached to switch A or B depending on whether they have been identified as a client or a server. Both switches mirror their outgoing traffic to switch C where all traffic is captured and finally exported as the result of the generation process. Mininet uses virtual Ethernet pair to emulate network link and we do not set any constraints(bandwidth, latency,..) on them for the accuracy of the IPT. In our experiments (see below), the virtual links are able to transfer data at a rate of at least 20GB/s.

5.5. Virtual hosts

Once a virtual host is created and configured by the NetworkHandler, it awaits instructions on how and when to send traffic to other hosts. As explained in Section 5.3, instructions include information about the destination of the traffic, the start time of the first packet to send, the IPTs, and the PSs.

If the flow is a TCP flow, the host identified as the initiator of the communication is responsible for opening the TCP connection to the destination, given that the connection has not been already opened in a previous time frame.

We want to emphasize that except for the first packet of a time frame, there is no synchronization between two hosts to decide whose turn it is to send a packet. Each host will send packets according to its list of PS and IPT values generated by the model. We motivate this decision by the fact that we are interested in evaluating flow-based IDS, meaning that we aim to only replicate flow level information from the input trace and not application level behavior. A possible way to synchronize bidirectional packet exchanges using TCP sequence numbers was proposed in [23]. However, it would require to retain much more information (such as the sequence numbers in both direction) from the original trace, while in our approach only the PS and IPT values need to be kept.

Since our traffic generator works on packet level and not on application level, the hosts fill packets with random content. However, TCP being a stream oriented protocol, there is no guarantee that blocks of data sent through several `send` operations on the BSD socket will be sent as individual packets. Similarly, the receive operation at the receiver socket might treat the payload of multiple packets as one single big block. This would result in communication patterns that are very different from the input trace. To have strict control on the

number and the size of exchanged packets, we disable Nagle's algorithm [36] on all hosts. This algorithm is used in TCP to reduce the number of packet sent in the network by merging small packets into bigger ones. Furthermore, we include a four-byte value in each sent packet that indicates to the receiver how many bytes it should read at once from the stream.

5.6. Generating attack traffic

Since the goal is to evaluate IDS, the generated trace must include malicious traffic. In order to simulate attacks, the user has to provide a configuration file containing the start time of the attack, the IP address of the attacker and the target and a script file executing the attack. When the FlowHandler has reached the time frame corresponding to the start time of the attack, it will notify the NetworkHandler which then creates the virtual host for the attacker (if it does not already exist) and instructs it to execute the attack script. The script contains the code that will perform the attack.

Any network attack that does not require information about application protocols can be performed as long as the script is provided. Typically traffic of attacks such as SYN Flooding, TCP Scan or TCP hijacking can be generated. As mentioned earlier, since the hosts are emulated, they will react to malicious traffic. For example, during a TCP Scan, the target victim sends RST packet if the destination port of the attack SYN packet is not open.

Only generating network attacks is an obvious limitation of this approach (shared with all generators not operating on application level). Since we do not simulate any application software on the hosts, a packet containing, e.g., a buffer overflow attack would not have any impact on the emulation. Nevertheless, it can be detected if it is sufficiently different from legitimate traffic in terms of packet statistics, as shown in Section 6.4. Moreover, network attacks, such as scans or flooding, also occur in ICS, for example during the propagation of a worm.

6. Experimental Validation

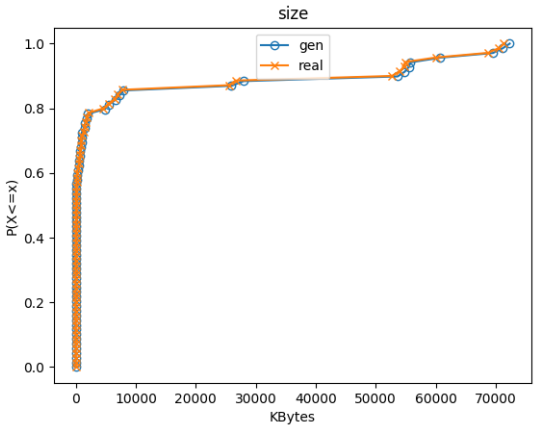
Our experimental validation consists of two parts. First, in Section 6.1, we evaluate the characteristics of the traces generated by our generator without malicious traffic. We evaluate our trace generator on three input datasets. Second, we study the practical usefulness of our generator in Sections 6.2 through 6.7 for the evaluation of different IDS proposed in the literature. Some of them are specifically designed for ICS. In the original papers, they were evaluated on control systems communicating with emulated field devices.

All experiments were run in the virtual machine provided by the Mininet project on a Dell XPS13 with 4GB of RAM.

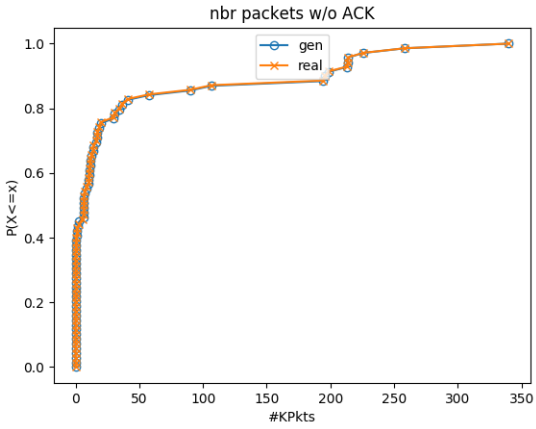
6.1. Benign trace generation

6.1.1. Experiments with our BMS trace

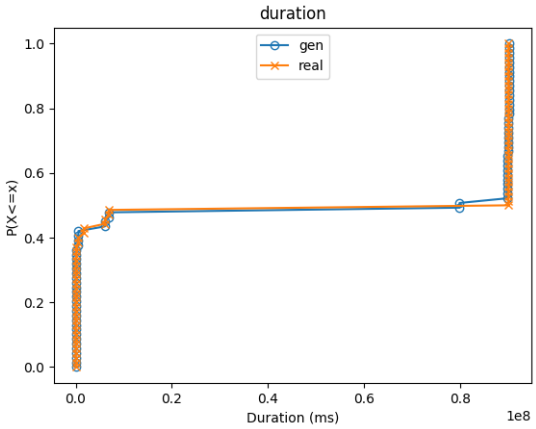
The first input trace is our BMS trace described in Section 4. We generate a one-day trace from it. The duration of a time frame is one hour. In the



(a) CDF of the number of bytes per flow



(b) CDF of the number of packets per flow



(c) CDF of the flow duration

Figure 5: Comparison between generated and empirical flows

following, we focus on the communication between the control server and the gateways on port 2499 and the communication between the HMI and the control server on port 50000 and 135, which are the HVAC specific ports. This trace does not contain malicious packets.

We compare the flow-level characteristics of the input trace and the generated one. Figure 5a shows the flow-size distributions for both traces. As can be seen, the generator is able to mimic the distributions with an acceptable level of accuracy. It is important to mention that this comparison is computed only from the packet payload sizes. The headers are not taken into account. We can also see that the generated flows are slightly larger than the original flows. This is caused by the added four-byte marker (see Section 5.5). Figure 5b compares the number of packets observed per flow. Again, the numbers are close to the

	Empirical	Generated
Mean PS (bytes)	78.352	82.359
Std PS	5.315	5.288
Mean IPT (ms)	264.169	265.03
Std IPT	397.846	398.789

Table 2: Comparison of the packet size (PS) and the inter-packet time (IPT)

original values.

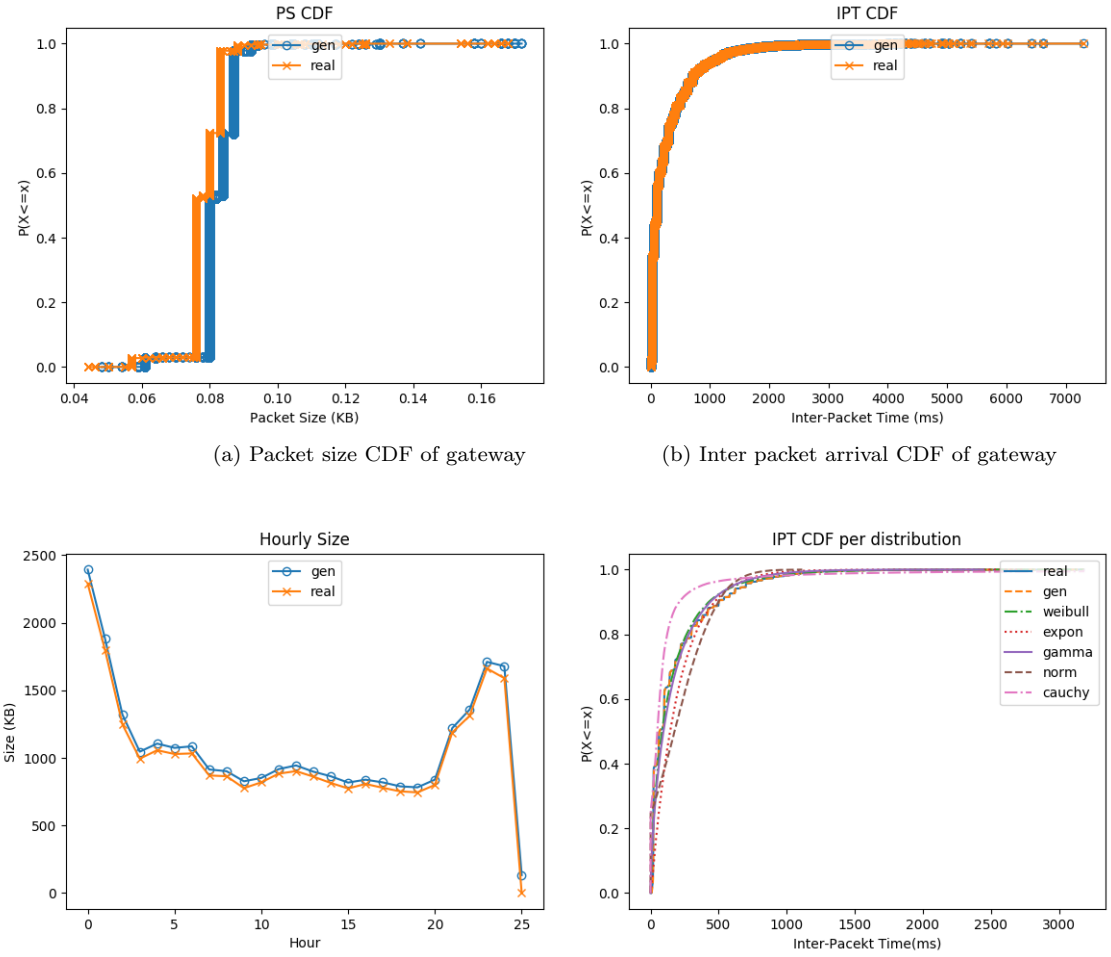
Figure 5c compares the duration of the flows. We observe that some flows take more time than their empirical counterparts while some take less time but the overall duration in the generated trace is close to the original.

Note that we have not taken TCP ACK packets into account in the above results since our goal is to obtain similar but not identical traces and not to replicate precisely every aspect of the original trace. We observe that less ACK packets are generated, though.

Then, we focus on the packet-level characteristics. We are particularly interested in a flow from one of the gateways to the control server because such flows have shown interesting variations over time in the original trace. Table 2 compares mean and standard deviation of the PS and IPT for this particular flow. The corresponding CDF are shown in Figure 6a and 6b. As can be seen, the error is small (again, note the shift by four bytes caused by the added four-byte marker). Since we have chosen a time frame duration of one hour, the traffic generator recomputes the distribution of the PS and IPT every 60 minutes during the trace generation and so it is able to track the time dependency of the gateway's activity. This is visible in Figure 6c, which shows the number of bytes per hour sent by the gateway to the control server.

Then, we take a two-hour trace of the communication between the control server and the gateway and we use it to generate two hours of legitimate traffic with our generator. We compare the IPT of the real trace, our generated trace and traces generated with D-ITG [23]. Figure 6d shows the CDF of the IPT observed for our traces and the different distributions offered by D-ITG. Visibly, the trace generated by our generator (labeled "gen" in the plot) matches best the real trace. Another advantage of our generator is the fact that the generated traffic is bidirectional. Having bidirectional flows is more realistic with regard to how hosts communicate in ICS, which is necessary for a realistic evaluation of IDS (see Section 6.7).

Since our trace generator estimates PS and IPT distributions per (relatively large) time frames, it is not able to correctly reproduce short-range dependencies between packets. We have calculated the Auto-Correlation Coefficient (ACC) of the PS and IPT for the flows in the real trace and found ACC values relatively evenly distributed between -0.2 and +0.8 (PS; Figure 7) and between -0.8 and +0.8 (IPT; Figure 8). In contrast, since our generator draws samples randomly from the learned distributions, it produces uncorrelated series of PS and IPT, resulting in ACCs close to zero. This could be fixed by using models that cor-



(a) Packet size CDF of gateway

(b) Inter packet arrival CDF of gateway

(c) Bytes sent by the gateway

(d) CDF IPT of 2h trace w/ gen. and D-ITG distributions

Figure 6: Characteristics of the flow between a gateway and the control server

rectly estimate short-range dependencies. However, we argue that this should not matter much for flow-based IDS, which are the IDS we are primarily interested in. Flow-based IDS mostly base their detection on information collected from aggregated measurements. Furthermore, generating traces which behave similarly flow-wise but with differences in how the packets are exchanged can help to assess how sensitive an IDS is to small packet-level deviations.

6.1.2. Experiments with a testbed trace

The next input trace has been generated from a public dataset collected at a testbed [12]. The dataset contains Modbus/RTU messages exchanged between a Master Terminal Unit (MTU) and Remote Terminal Unit (RTU) in a gas pipeline control system testbed. The dataset is provided as a text file

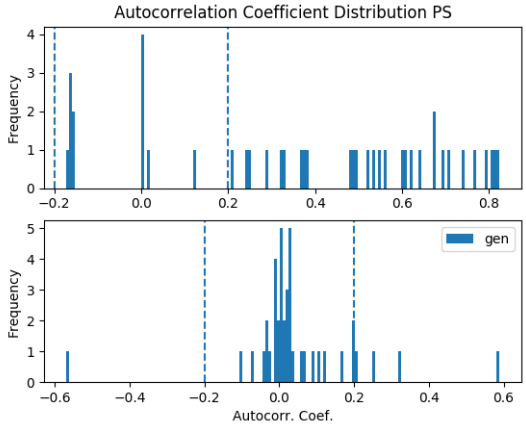


Figure 7: Distribution of the PS auto-correlation coefficient. Top: Real trace, Bottom: Generated trace.

with the payload of the messages and their respective timestamp. We only keep legitimate messages and convert them to a packet trace by replaying the payloads in Modbus/TCP packets. For each packet in the obtained trace, we set its timestamp to the one in the original dataset. We generate a one-hour trace. Figures 9a and 9b show that the generated trace matches well the original trace.

6.1.3. Experiments with a non-public real SCADA trace

Finally, we have applied our generator to a SCADA trace from a utility company. The trace contains the traffic exchanged between MTUs and several dozen PLCs. Unfortunately, due to the confidential nature of the trace, we cannot provide further details. Again, we generate a one-hour trace. Figures 10a and 10b compare the PS and IPT CDFs of the generated trace with the original trace.

6.2. Generation of benign and malicious traffic

After evaluating the realism of the generated traces, we want to measure their usability for evaluating flow-based IDS. The goal of these experiments is to show how the traces can be used to evaluate the detection performance of IDS and that the IDS do not exhibit unexpected behaviors when used with the generated traces, i.e. that the generated traces look "realistic" to them. To this end, we select several IDS from the literature, some of them explicitly designed for ICS environments, some not. We focus more on having a diverse pool of IDS than having IDS with good detection performance.

We generate traces which include both legitimate and malicious traffic. The legitimate traffic is generated from our BMS trace and malicious traffic comes from two kinds of scenarios. The first scenario represents an attacker trying to obtain information about the ICS by performing horizontal and vertical scans.

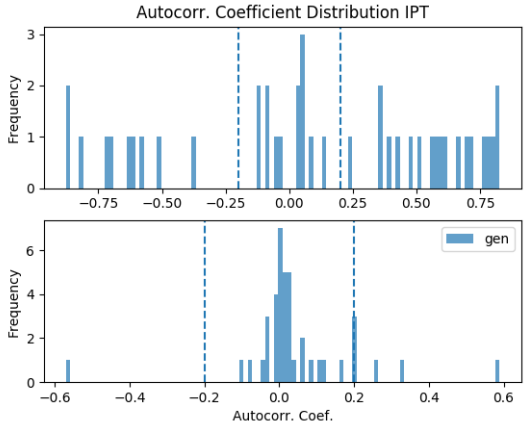


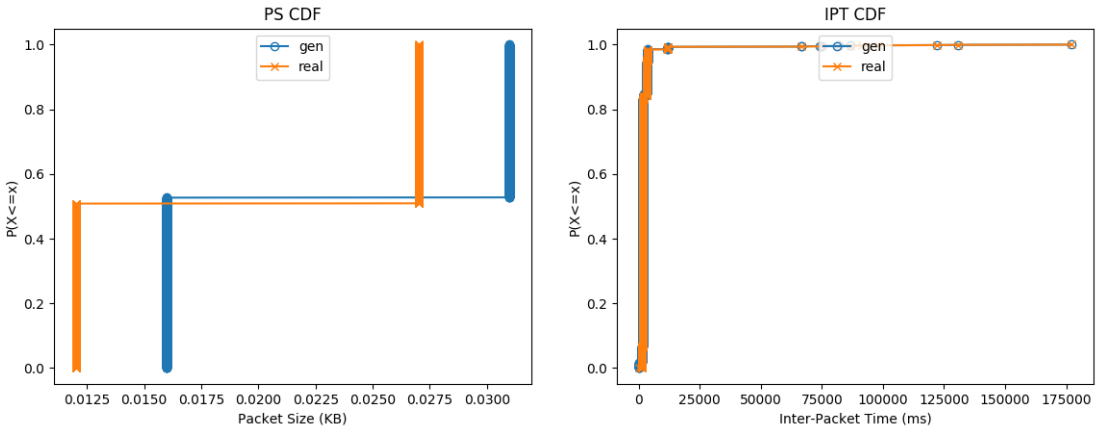
Figure 8: Distribution of the IPT Auto-correlation coefficient. Top: Real trace, Bottom: Generated trace.

IDS	Scan H.	Scan V.	TCP Seq. Enum.
SCADA Pattern-based [38]	Yes	Yes	No
SCADA Flow-based [38]	No	No	Yes
EWMA-based [39]	Yes	Yes	No
Sketch-based [40]	Yes	Yes	Yes
Entropy-based [41]	No	Yes	Yes

Table 3: List of evaluated IDS and attack detection

In this scenario, an attacker looks for hosts in a range of IP addresses (horizontal scan) and upon finding a host, the attacker scans it for open ports (scan vertical). The second scenario represents an attacker trying to inject a malicious packet into the communication between one of the gateways and the control server by spoofing the IP address of the control server and guessing the correct TCP sequence number [37]. The attacker sends one packet per receiving window hoping to find the correct sequence number range. This kind of attack has more chance to work on older systems and with long connections, so it is safe to assume that it could happen in an ICS-like environment.

We have chosen these scenarios because of the different effects they cause on the network. The first scenario increases the number of flows in the ICS while the second scenario increases the number of packets in a flow. All the attacks were run against every IDS that are presented in this paper but we only discuss the interesting results because some of the IDS can, by design, not detect all types of attacks. Table 3 lists the IDS and their detection capabilities with regard to the considered attacks.



(a) Packet Size CDF

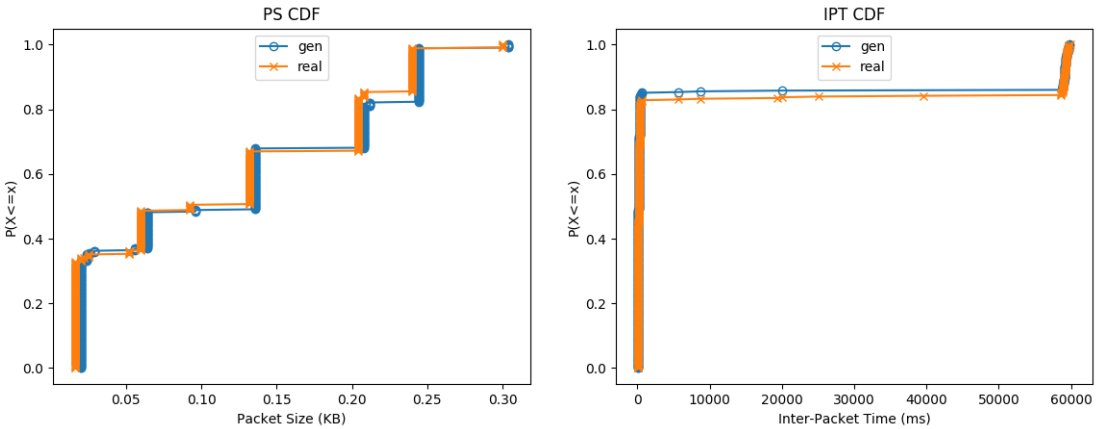
(b) Inter packet arrival CDF

Figure 9: Results for the testbed trace

6.3. Evaluation of a pattern-based IDS

The first IDS proposed in [38] maintains a library of patterns for each host. A pattern is the set of IP addresses with which a host has communicated during a time window and the normalized number of packets exchanged with each IP. The library of a host is first empty and periodically updated every T seconds. After each period, a new pattern is created based on the network activities of the host in the past T seconds and that new pattern is compared with the existing patterns in the library. If a library pattern is similar to the new pattern, the former is updated. Otherwise, the new pattern is added to the library. The similarity test is controlled by a threshold value that we have set to 0.7 according to [38]. For each pattern, the IDS counts how often the pattern has been observed. The IDS raises an alarm if the tail probability of a pattern is lower than a predefined alert threshold θ.

We generate a two-hour trace where an attack is performed after around 45 minutes. The legitimate traffic is generated from our empirical BMS trace. The attack is a scan done both horizontally and vertically. The period size is set to 180 seconds and $\theta = 0.5$. The attack starts at the end of the 13th detection period and runs until the end of the generation. The IDS detects the attack in the 15th period and all following ones. Figure 11 shows for the control server how many packets it has exchanged with each host (with host i being the attacker) before the attack (left) and during the attack (right). We repeat the trace generation and test the IDS also with a period size of 15 seconds, the attack starting after around 20 minutes, and other threshold values $\theta \in \{0.1, 0.2, 0.3, 0.4\}$. In all test runs, the IDS correctly identifies the attacking host and no false positive alarms are raised. This shows that our generator is able to produce traces that look "natural" to the tested IDS and that do not



(a) Packet size CDF

(b) Inter Packet arrival CDF

Figure 10: Results for the non-public SCADA trace

break any assumptions made by the IDS designers about the traffic.

As for the enumeration attack, the IDS is unable to detect it. Since gateways only communicate with the control server, the attacker bypasses the detection by spoofing the IP address of the control server when talking to the gateway.

6.4. Evaluation of a flow-based IDS

The second IDS proposed in [38] is a flow-based IDS. The authors define a flow by the source IP address, destination IP address and destination port. Notice that this flow definition does not consider the source port. The IDS computes flow information periodically and a database storing flow information over several periods is used to detect attacks. For each flow and observation period, the IDS computes the number of packets, the average and the variance of PS and IPT. At the end of an observation period and when the packet count is large enough, the IDS runs a T-test for both PS and IPT. The result of the test determines whether an alarm is raised.

The legitimate traffic is generated from our BMS trace. We use our generator to simulate the enumeration attack described in 6.2.

We set the period size of the detection process to 300s. The IDS needs to see flow information for a fixed number of periods to consolidate the record of historical flows before entering the detection phase. We set this number to 12. The attack starts in the middle of the 8th period and it is detected immediately when the IDS enters the detection phase. The attack impacts the PS and IPT statistics of the flow between the control server and the gateway. The PS average increases because the malicious packet (156 bytes) the attacker is trying to inject is bigger than packets observed under normal condition. Correspondingly, the average IPT decreases due to the rate at which the attacker is sending the packet

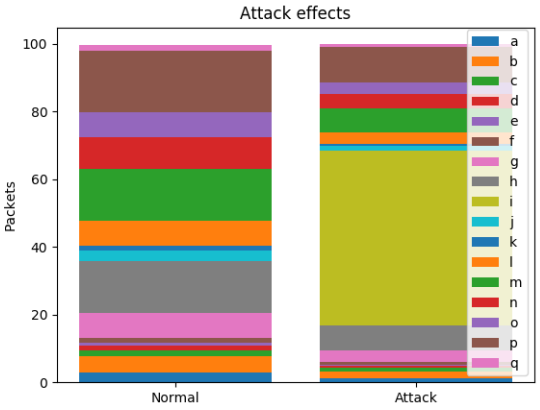


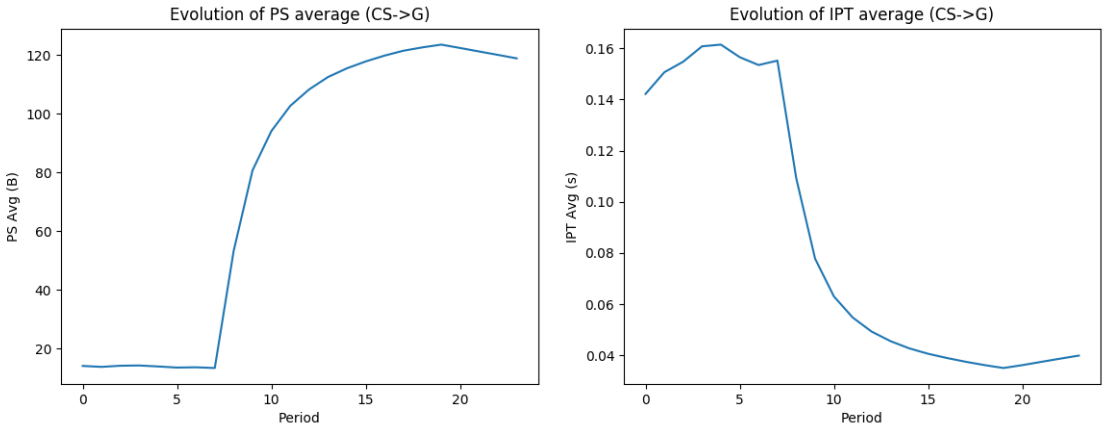
Figure 11: Number of packets exchanged by the control server with other hosts. Host i is the attacker performing a port scan of the control server.

(approx. 1 packet every 50ms). The effect of the attack over time is shown in Figure 12. The reverse direction of the communication, i.e. the flow from the gateway to the control server is not affected. The attack is not successful so the gateway simply ignores the packets.

While the flow-based IDS is able to detect the attack in our test trace, it also raises false alarms. In total, the IDS generated 78 alarms for 52 flows whereof only 12 alarms were true positives. On average, 6.5 alarms were raised per period (without taking into account the learning periods). As explained, the IDS uses the T-test to determine if the average PS (resp. IPT) of a flow in the current period is similar to the historical average for that flow. The T-test compares a T-score $t_{calculated}$ computed from the flow statistics with a predefined standard value t_{value}. If the $t_{calculated}$ is greater than the t_{value} then the test has failed. An alarm is generated when at least one of the T-tests is rejected.

Obviously, the T-test is too strict for the generated trace. Even worse, when applied to our empirical, attack-free trace, the IDS raises 82 alerts. To reduce the number of false positive, we add a margin m such that an alert is raised when the T-score $t_{calculated}$ is greater than $t_{value} + m$. The margin m is computed in the following way: For each period, we take the average of the differences between the t_{value} and $t_{calculated}$ when there is an alert. The margin is then set to the maximum of those averages. Note that m is computed for both the PS and the IPT and for an attack-free trace. To evaluate the impact of m, we compare the number of alerts raised on the empirical trace (REAL), a generated attack-free trace (GEN) and a generated trace containing an attack (ATK). For each of the trace, we consider three ways to determine m for the PS and the IPT:

- m is set to 0 for PS and IPT;



(a) Packet size average

(b) Inter-packet time average

Figure 12: Impact of injected attack packets on the statistics of the flow from the control server to a gateway

- m is calculated for PS and IPT according to the method described above, using the REAL trace;
- m is calculated for PS and IPT according to the method described above, using the GEN trace.

Figure 13 shows the resulting number of alerts raised by the IDS when monitoring the three traces REAL, GEN, and ATK using the three ways to determinate m. As expected the number of alerts is the highest when m is set to 0. When the IDS is applied to the ATK trace, the number of alerts raised is exactly the same when m is either computed from GEN or from REAL, without a negative impact on the number of true positives (not shown here). However, this is not the case for the other traces: When the IDS analyzes the REAL and GEN traces, the number of (false) alarms is less when m is computed from GEN. In this experiment, we observed that the margin for the PS (resp. IPT) computed from REAL was 6.067 (resp. 6.075) while it was 8.58 (resp. 5.66) when computed from GEN.

Effectively, this means that our generated traces have helped us to find a better value for the parameter m of the detection process. This illustrates the usefulness of the generator; the generated traces allow testing IDS under slightly different but still realistic conditions and therefore can help to find more suitable parameter values.

The IDS is not able to detect the scan attacks because the flow generated by them are too short lived. The number of packets is too small to be able to perform the T-Test. All the alarms generated were false positives.

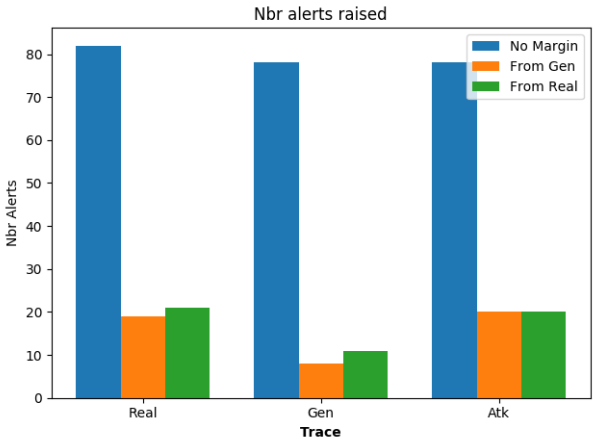


Figure 13: Number of alerts raised by the the flow-based IDS on different traces without error margins resp. with error margins computed from two attack-free traces.

6.5. Evaluation of a flow-based IDS for non-ICS networks using forecasting

The two IDS evaluated previously were designed for ICS-like traffic. We analyze in this and the following two sections three IDS that were not designed for ICS [39, 40, 41]. The two first IDS detect attacks by predicting the evolution of some flow metric (e.g number of flows seen) and comparing the prediction with real observations. The last one uses the entropy of the packet distribution to detect attacks. To evaluate these IDS, we use five traces: a two-hour long attack-free trace from the real BMS (labeled *trace1* in the following) and four two-hour long generated traces containing horizontal and vertical scans (labeled *trace2* through *trace5*).

The first IDS we evaluate leverages the Exponential Weighted Moving Average forecasting technique (EWMA) to detect anomalies in flow metrics. We only consider the basic algorithm described in [39]. The IDS divides the time into intervals and predicts the next value

$$\tilde{x}_{t+1} = \alpha \cdot x_t + (1 - \alpha) \cdot \tilde{x}_{t-1}$$

of an observed metric from the current observation x_t and prediction $\tilde{x}_{t-1}$, where α is the weight of the moving average. Instead of directly comparing the forecasting error $e_t = x_t - \tilde{x}_t$ to a threshold, which could lead to many false positives, the authors calculate a cumulative sum

$$S_t = max(S_{t-1} + (x_t - T_{upper,t}), 0)$$

based on x_t and $T_{upper,t} = \tilde{x}_t + max(c_{threshold} \cdot \sigma_{e,t}, M_{min})$, where $c_{threshold}$ is a constant and $\sigma_{e,t}$ is the standard deviation of previous forecasting errors. When the sum exceeds the threshold

$$T_{cumsum,t} = c_{cumsum} \cdot \sigma_{e,t}$$

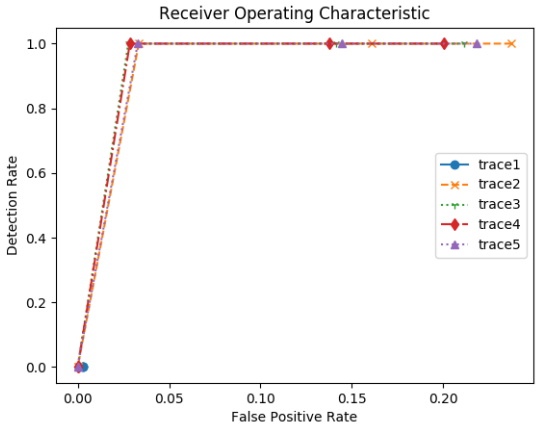
(where c_{cumsum} is a constant) an alarm is raised. In that case, the observed values are ignored so that the forecasting is not influenced by the abnormal values of the attack. In their evaluation, the authors look at the influence of the parameters $c_{threshold}$ and c_{cumsum} on the performance of the IDS. They show that c_{cumsum} has little impact, compared to $c_{threshold}$ which affects both the Detection Rate (DR) and the False Positive Rate (FPR). However, there is a an other parameter which can affect the IDS capability. The detection scheme of the IDS is to check if $S_t > T_{cumsum,t}$ in an interval t. During an attack, x_t increases and exceeds T_{upper} leading to an increase of S_t. However, when the attack finishes, S_t will still be higher than $T_{cumsum,t}$ for some intervals, meaning that for those intervals, alarms will be raised even though the attack is over. To avoid that, the authors mention an upper bound to S_t but they do not discuss its impact on the IDS. We call this upper bound S_{upper}.

We investigate the impact of S_{upper} on the Detection Rate and the FPR by modifying its value for different fixed M_{min} values. M_{min} is a margin used by the IDS to control the threshold in case $c_{threshold} \cdot \sigma_{e,t}$ is too small. The metric that the IDS forecasts is the number of new flows created per period. Note that a flow which lasts for several periods is not considered as a new flow.

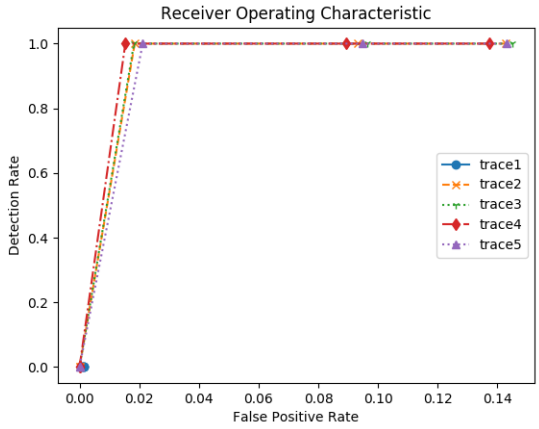
Figure 14 shows the Receiver Operating Characteristic of the IDS in experiments with $S_{upper} \in \{5, 15, 20, 50, 70\}$ for three different values of M_{min}. The other parameters are set to $c_{threshold} = 1.5$, $c_{cumsum} = 5$, and $\alpha = 0.01$. The interval size is set to 5 seconds. Several observations can be made. First, it is clear that S_{upper} affects greatly the FPR. A lower S_{upper} leads to a lower FPR. This is because S_t decreases faster after an attack which reduces the number of normal intervals for which S_t exceeds $T_{cumsum,t}$. However, if the upper bound is too low (e.g. 5 in our experiment), it might never exceed the threshold even during an attack meaning that attacks are never detected. This means there is a trade-off between DR and FPR. Second, when M_{min} is small (i.e. 0 or 5), the impact of S_{upper} on the FPR is not the same for the difference traces. This is caused by the random nature of the generation process. Although the generated traces are identical in terms of high-level statistics, such as the number of flows, packets may end up in different intervals, which impacts the forecast and since M_{min} is low, it is the forecasting error that influence the most T_{upper} and therefore S_t.

Contrariwise, when M_{min} is large ($= 20$) the impact is more uniform among traces. This can be explained by the fact that when M_{min} is large, $T_{upper,t}$ tends to be large too, so S_t increases slower. Therefore, S_t does not necessarily reach S_{upper}. This also explains why the FPR is low when M_{min} is large since S_t exceeds T_{cumsum} less often. In the case where M_{min} is low, the IDS can reach a DR of 100% with a FPR of 20%. In the other case, it can detect 97% of the attack without raising any alarms. The results show that the value of M_{min} influences how sensitive the IDS is to the order in which the packets are sent. A flow-based IDS uses aggregate metrics so one would expect that the order in which packets are exchanged should not matter much. However, thanks to the generated traces, we can see in Figure 14a that this is not always true.

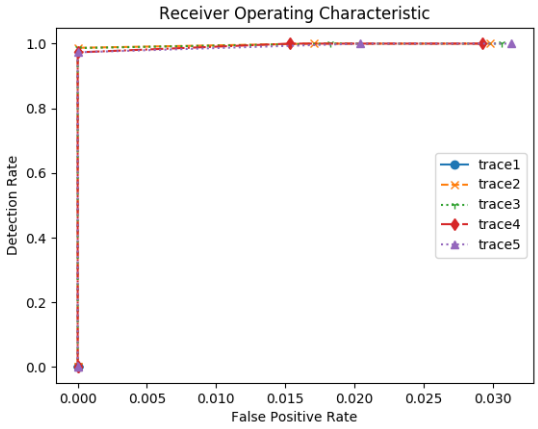
The IDS is not able to detect the enumeration attack since the attack does



(a) EWMA IDS, $M_{min} = 0$



(b) EWMA IDS, $M_{min} = 5$



(c) EWMA IDS, $M_{min} = 20$ (NB:x-axis scale)

Figure 14: IDS using EWMA forecasting with different M_{min} values

not impact the number of flows created.

6.6. Evaluation of a sketch-based IDS

The IDS in [40] relies on a K-ary Sketch. This is an array of L mutual independent hash tables of size C. The table keys are IP addresses and the values are user-defined counters (e.g number of packet received). The time is divided into intervals of duration T and at the end of each interval ($k \cdot T$ with k a positive integer) the IDS does the following: For each counter, the IDS uses the Least Mean Square (LMS) to forecast the value of the counter in the next interval from the n previous values. LMS attempts to minimize the error between observation and forecast by using a gradient-based method of steepest decent. The predicted value is computed from a weight vector W,

randomly initialized, and a vector of previous values. In each interval, LMS will correct W based on the error. At the end of an interval, for each hash table, the IDS computes two distributions: the forecast one and the observed one. The Chi-Square divergence is used to measure the distance between the two distributions. The IDS maintains for each hash table h two timeseries $\chi^2_{h,kT}$ and $\tilde{\chi}^2_{h,kT}$. $\chi^2_{h,kT}$ is the timeseries of the Chi-Square divergence computed up to the kT interval. $\tilde{\chi}^2_{h,kT}$ is a timeseries derived from $\chi^2_{h,kT}$ that does not contain the outliers from $\chi^2_{h,kT}$. To detect outliers, the IDS uses a threshold given by the Cheyseve inequality. A Chi-Square divergence d is considered an outlier if

$$d >= \mu_{h,(k-1)T} + \alpha \cdot \sigma_{h,(k-1)T},$$

where α is a constant and $\mu_{h,kT}$ and $\sigma_{h,kt}$ are the mean and the standard deviation of $\tilde{\chi}^2_{h,kT}$ respectively. They are updated dynamically with EWMA as

$$\begin{aligned} \mu_{h,kT} &= \beta\mu_{h,(k-1)T} + (1-\beta)\tilde{\chi}^2_{h,kT}, \\ \sigma^2_{h,kT} &= \beta\sigma^2_{h,(k-1)T} + (1-\beta)(\tilde{\chi}_{h,kT} - \mu_{i,kT})^2 \end{aligned}$$

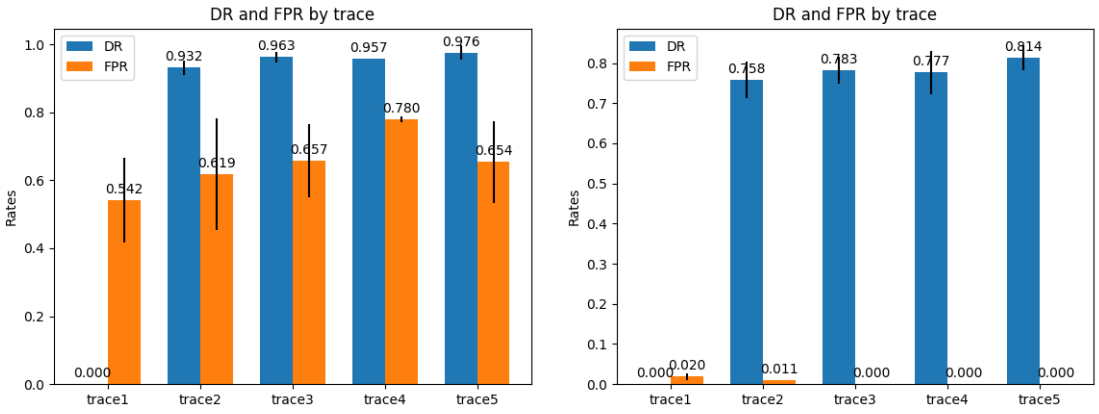
When an outlier has been detected, it will be replaced in $\tilde{\chi}^2_{h,kT}$ by the last value of $\tilde{\chi}^2_{h,(k-1)T}$. If outliers have been detected for more than H hash tables and for η consecutive periods an alarm is raised. Observed values are ignored for the forecasting during an attack.

We evaluate the IDS with the traces described in Section 6.5. The results are showed in Figure 15a. The parameters are fixed: $L = 5$, $C = 100$, $n = 5$, $H = 3$, $\eta = 1$, α=4, $\beta = 0.7$. As mentioned before, *trace1* is the real, attack-free trace and the others are generated traces containing attacks. The weight vector of the LMS techniques is initialized randomly so we perform 10 runs of the experiment. The metrics computed is the number of packets received by a host.

The IDS is able to detect attacks with an acceptable rate (above 90%). However, the FPR is high in every case. This can be explained by the update method of the $\tilde{\chi}^2_{h,kT}$ timeseries. When an outlier is computed in a period, the last non-outlier Chi-Square divergence value is used as a replacement. This means that if an attack lasts for several interval, the IDS will keep adding the same value to $\tilde{\chi}^2_{h,kT}$. This will decrease $\sigma_{h,kT}$ and therefore lead to a small threshold. A solution for this problem would be to use a lower bound for $\sigma_{h,kT}$. We apply this solution for lower bound of 0.2 and show the results in Figure 15a. We can see that while this solution reduces the detection rate, it drastically decreases the number of false positives.

Apart from *trace1*, which is attack-free, the performance of the IDS is quite similar for every generated trace suggesting that, in contrast to the previous IDS, the order in which packets are sent has little impact on the performance of the IDS.

We evaluate the IDS with the second scenario and we observe similar performance. The IDS has a Detection Rate of 100% and False Positive Rate of 17.4% when there is no lower bound for $\sigma_{h,kT}$. With the lower bound, the DR becomes 96.7% and the FPR becomes 0%.



(a) Unmodified IDS

(b) IDS with lower bound

Figure 15: Performance of the Sketch-based IDS

6.7. Evaluation of an entropy-based IDS

The last IDS evaluated relies on the PS distribution inside flows [41]. The motivation is that during a flooding attack, many small packets of the same size will be sent to the target host, leading to less diversity in the PS distribution. The authors define a flow as a 2-tuple (sip_i, dip_i) where sip_i is the source IP address and dip_i destination IP. Before running, the range of expected packet sizes must be defined. Then this range is manually divided into N bins of various size (e.g $[40, 60]$, $[60, 80]$, $[80, 120]$,...). Using these bins, the IDS computes an entropy score from the observed packet sizes:

$$H(dip) = -\sqrt{i_{max}} \cdot \sum_{i=1}^{N} p(x_i) log_2 p(x_i) (1 \leq i \leq N),$$

where $p(x_i)$ is the probability that the packet received by dip falls in the i^{th} bin and i_{max} is the bin with the highest probability. During an attack, the packet sizes will gather in lower packet size bins leading to a lower entropy score. The score is computed every T period and an alarm is raised if it is less than a threshold θ.

Applying the IDS directly to our traces results in an FPR of 100%. The reason for this is the fact that some hosts always receive packets of the same size, a typical behavior for control system networks. In order to make the IDS more suitable to our scenario, we ignore flows with an entropy score of 0 and assume that attack packets will differ sufficiently from regular packets to raise the entropy above 0 during attacks. We have determined empirically that the best results are obtained with many small bins for small packet sizes. For the experiments, we divided the range $[0, 20[$ into 10 bins of 2 bytes, the range

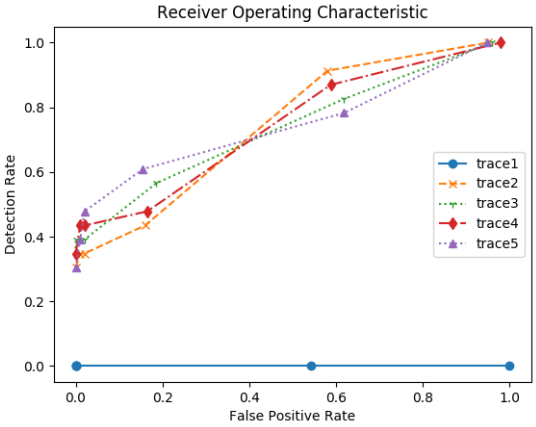


Figure 16: TrustGuard ROC

$[20, 200[$ into 10 bins of 10 bytes, the range $[200, 1000[$ into 20 bins of 50 bytes and finally the range $[1000, 1600]$ into 3 bins of 200 bytes. The period size was set to 60 seconds.

Figure 16 shows the Receiver Operation Characteristic for $\theta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.8\}$. It shows that higher thresholds increase both DR and FPR for all traces. Again, this can be explained by the regular nature of communication in control systems. The higher θ, the more likely that attack-free flows with packet sizes of low variability will be classified as attacks. This IDS is a good example of a solution designed for traditional ICT network that is not suited for control and automation systems. The results show that even though the overall evolution of the rates looks similar for the different traces, there are significant differences in term of performance depending on the trace. For example, with a threshold of 0.4, the IDS reaches a DR of 60% with *trace5* while it only detects 40% of the attack *trace2*. This shows that an IDS that is *per se* designed to only look at PS distributions can still be considerably sensitive to differences in packet timing and order. Note that the results would be even worse if our trace generator was not generating bidirectional traffic flows. In a unidirectional generator, only one direction carries traffic while the other mostly consists of empty ACK packets of identical size, resulting in an entropy of zero.

As far as the TCP sequence number enumeration attack, the IDS is not able to detect it when the threshold θ is below 0.4. In our scenario, the probes sent by the attacker have varying size meaning that the entropy does not decrease enough so that the IDS can detect the attack. With a threshold of 0.4, the IDS has DR of approx. 18% and a FPR of approx. 11%. Similarly to the result obtained with the scans attacks, a higher threshold leads to a better DR but also a higher FPR.

6.8. Practical performance and limitation

Our current implementation of the generator runs on a single machine. Given the characteristics of ICS, this is not big issue. The low throughput allows to estimate distributions on the fly and the presence of long-live flows reduces the overhead to open and close connections in the emulated network. Moreover, the stability of the ICS network topology causes that most of the virtual hosts are created at the beginning of the trace generation and, once they are created, changes to the emulated network are rare.

According to the classification in [19], our generator can be seen as a hybrid flow/packet-level generator. It replicates PS and IPT distributions but it leaves the creation of the packets to Mininet. On one hand, this means that some information present in the original trace that could be useful for intrusion detection, like TCP/IP fingerprinting, is lost. On the other hand, this allows users to experiment with other protocols (e.g. IPv6 instead of IPv4) by changing the configuration of Mininet. Since the scope of this paper is the evaluation of flow-based IDS, we have not discussed this point further.

7. Conclusion

Research conducted on ICS has showed that the characteristics of their network communication differentiate them from traditional IT networks to the point that researchers have designed ICS-specific Network Intrusion Detection Systems (N-IDS). Despite the large number of papers published on IDS for ICS, the state of the art in the evaluation of new detection algorithms is more than unsatisfactory: Nearly all existing publications rely directly or indirectly on experiments in testbeds or simulated environments. Those few that use real ICS have not published their datasets.

The contribution of our paper to improve this situation is two fold: We have described and published a network trace collected in the building management system (BMS) of a university campus. BMS depict common characteristics of automation systems, such as long-lived connection, while also depicting diurnal patterns often associated with Internet traffic. To our knowledge, this is the first public trace of its kind. It can be downloaded from

https://github.com/hvacanon/HVAC-ANON

Our second contribution is a traffic generator framework and tool that can be used to generate traffic traces for the evaluation of flow-based IDS. The tool takes as input a network trace and replicates its flow-level statistics as well as certain packet-level statistics. It supports the generation of benign as well as malicious traffic. We have provided experimental results to show the ability of the generator to correctly replicate various existing traces. We have demonstrated the usefulness of the generator by evaluating five IDS found in the literature. Among others, our results show that the generator can help understanding and adjusting the sensitivity of detection algorithms.

References

[1] E. J. M. Colbert, S. Hutchinson, Intrusion Detection in Industrial Control Systems, Springer International Publishing, Cham, 2016, pp. 209–237.

[2] C. S. et al., Using model-based intrusion detection for scada networks, in: Proceedings of the SCADA Security Scientific Symposium, 2007.

[3] S. A. et al., An overview of ip flow-based intrusion detection, IEEE Communications Surveys Tutorials 12 (3) (2010) 343–356.

[4] K. R. et al, Traffic measurement and analysis of building automation and control networks, in: Dependable Networks and Services, Springer, 2012, pp. 62–73.

[5] R. R. R. B. et al, Flow whitelisting in scada networks, International Journal of Critical Infrastructure Protection 6 (3) (2013) 150–158.

[6] Z. Zheng, A. L. N. Reddy, Safeguarding building automation networks: The-driven anomaly detector based on traffic analysis, in: 26th International Conference on Computer Communication and Networks (ICCCN), 2017, pp. 1–11.

[7] M. Katagi, S. Moriai, et al., Lightweight cryptography for the internet of things, Sony Corporation 2008 (2008) 7–10.

[8] K. A. McKay, L. Bassham, M. S. Turan, N. Mouha, Report on lightweight cryptography (2017).

[9] Capture files from 4sics geek lounge 2015, `https://www.netresec.com/index.ashx?page=PCAP4SICS` (accessed: October 13, 2018).

[10] A. Lemay, J. Fernandez, Providing scada network data sets for intrusion detection research, in: 9th USENIX Workshop on Cybersecurity Experimentation and Test (CSET'16), 2016.

[11] R. M. et al, Flow-based benchmark data sets for intrusion detection, in: European Conference on Cyber Warfare and Security, 2017, pp. 361–369.

[12] T. Z. T. I. Morris, T., Industrial control system simulation and data logging for intrusion detection system research, in: 7th Annual Southeastern Cyber Security Summit, 2015.

[13] G. Kabasele Ndonda, R. Sadre, A public network trace of a control and automation system, arXiv preprint arXiv:1908.02118 (2019).

[14] A. E. E. Byres, J. Carter, D. Hoffman, Worlds in collision: Ethernet on the plant floor, in: ISA Emerging Technologies Conference, Instrumentation Systems and Automation Society, 2002.

[15] Caida (accessed: 2020-03-30).
URL https://www.caida.org/data/overview/

[16] J. Cao, W. S. Cleveland, , K. Jeffay, F. D. Smith, M. Weigle, Stochastic models for generating synthetic http source traffic, in: IEEE INFOCOM 2004, Vol. 3, 2004, pp. 1546–1557 vol.3.

[17] J. Sommers, P. Barford, Self-configuring network traffic generation, in: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, ACM, 2004, pp. 68–81.

[18] P. Kamath, , J. Heidemann, J. Bannister, J. Touch, Generation of high bandwidth network traffic traces, in: Proceedings. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002, pp. 401–410.

[19] A. Botta, A. Dainotti, A. Pescape, Do you trust your software-based traffic generator?, IEEE Communications Magazine 48 (9) (2010) 158–165.

[20] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, F. D. Smith, Tmix: a tool for generating realistic tcp application workloads in ns-2, ACM SIGCOMM Computer Communication Review 36 (3) (2006) 65–76.

[21] A. Klassen, Fred, Tcpreplay - pcap editing and replaying utilities (2018, accessed: 2020-03-30).
URL https://tcpreplay.appneta.com/

[22] iperf - the ultimate speed test tool for tcp, udp and sctp (accessed: 2020-03-30).
URL https://iperf.fr/

[23] A. Botta, A. Dainotti, A. Pescapé, A tool for the generation of realistic network workload for emerging networking scenarios, Computer Networks 56 (15) (2012) 3531–3547.

[24] A. Sperotto, R. Sadre, P.-T. de Boer, A. Pras, Hidden markov model modeling of ssh brute-force attacks, in: Integrated Management of Systems, Services, Processes and People in IT, Springer Berlin Heidelberg, 2009, pp. 164–176.

[25] J. Sommers, V. Yegneswaran, P. Barford, A framework for malicious workload generation, in: 4th ACM SIGCOMM conference on Internet measurement, ACM, 2004, pp. 82–87.

[26] J. Sommers, V. Yegneswaran, P. Barford, Toward comprehensive traffic generation for online ids evaluation, Tech. rep., University of Wisconsin-Madison Department of Computer Sciences (2005).

[27] R. Al-Dalky, O. Abduljaleel, K. Salah, H. Otrok, M. Al-Qutayri, A modbus traffic generator for evaluating the security of scada systems, in: 2014 9th International Symposium on Communication Systems, Networks Digital Sign (CSNDSP), 2014, pp. 809–814.

[28] T. Morris, A. Srivastava, B. Reaves, W. Gao, K. Pavurapu, R. Reddi, A control system testbed to validate critical infrastructure protection concepts, International Journal of Critical Infrastructure Protection 4 (2011) 88–103.

[29] Modbus messaging on tcp/ip implementation guide v1.0b (2006, accessed on 2020-03-30).
URL http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[30] S. K. A. et al., Sp 800-82. guide to industrial control systems (ics) security: Supervisory control and data acquisition (scada) systems, distributed control systems (dcs), and other control system configurations such as programmable logic controllers (plc), Tech. rep. (2011).

[31] C. B. et al., Specification of the ip flow information export (ipfix) protocol for the exchange of flow information, STD 77, RFC Editor (2013, accessed: 2020-03-30).
URL http://www.rfc-editor.org/rfc/rfc7011.txt

[32] A. M. Khandker, P. Honeyman, T. J. Teorey, Performance of dce rpc, in: Second International Workshop on Services in Distributed and Networked Environments, 1995, pp. 2–10.

[33] R. R. R. Barbosa, R. Sadre, A. Pras, Difficulties in modeling scada traffic: A comparative analysis, in: Passive and Active Measurement, 2012, pp. 126–135.

[34] B. Lantz, B. Heller, N. McKeown, A network in a laptop: Rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, ACM, 2010, pp. 19:1–19:6.

[35] S. J. Sheather, Density estimation, Statistical science (2004) 588–597.

[36] J. Nagle, Congestion control in ip/tcp internetworks, RFC 896, RFC Editor (January 1984).

[37] ikm@phrack, Blind TCP/IP hijacking is still alive (2007, accessed: 2020-03-30).
URL http://phrack.org/issues/64/13.html

[38] A. Valdes, S. Cheung, Communication pattern anomaly detection in process control systems, in: 2009 IEEE Conference on Technologies for Homeland Security, 2009, pp. 22–29. doi:10.1109/THS.2009.5168010.

[39] R. Hofstede, V. Bartoš, A. Sperotto, A. Pras, Towards real-time intrusion detection for netflow and ipfix, in: Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013), 2013, pp. 227–234.

[40] A. Makke, O. Salem, M. Assaad, H. Moungla, A. Mehaoua, Flooding attacks detection in backbone traffic using power divergence, in: Proceedings of the 7th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, PM2HW2N '12, ACM, 2012, pp. 15–20.

[41] H. Liu, Y. Sun, V. C. Valgenti, M. S. Kim, Trustguard: A flow-level reputation-based ddos defense system, in: 2011 IEEE Consumer Communications and Networking Conference (CCNC), 2011, pp. 287–291.