TSNZeek: An Open-source Intrusion Detection System for IEEE 802.1 Time-sensitive Networking

Doğanalp Ergenç, Robin Schenderlein, Mathias Fischer
University of Hamburg, Germany
name.surname@uni-hamburg.de

***Abstract*—IEEE 802.1 Time-sensitive Networking (TSN) standards are envisioned to replace legacy network protocols in critical domains to ensure reliable and deterministic communication over off-the-shelf Ethernet equipment. However, they lack security countermeasures and can even impose new attack vectors that may lead to hazardous consequences. This paper presents the first open-source security monitoring and intrusion detection mechanism, TSNZeek, for IEEE 802.1 TSN protocols. We extend an existing monitoring tool, Zeek, with a new packet parsing grammar to process TSN data traffic and a rule-based attack detection engine for TSN-specific threats. We also discuss various security-related configuration and design aspects for IEEE 802.1 TSN monitoring. Our experiments show that TSNZeek causes only ∼5% CPU overhead on top of Zeek and successfully detects various threats in a real TSN testbed.**

***Index Terms*—intrusion detection, security, IEEE 802.1 TSN**

I. Introduction

Modern mission-critical systems are composed of several interconnected components and services that require reliable and time-sensitive communication To satisfy such requirements and reduce the dependency on domain-specific networking technologies, IEEE 802.1 Time-sensitive Networking (TSN) task group has proposed a set of standards. These standards enable low-latency, fault-tolerant, and deterministic communication on top of standard Ethernet protocols. However, IEEE 802.1 TSN protocols also induce several security threats across the domains and systems deploying them [1]–[3]. Timely detection of such threats is crucial, especially in safety-critical systems, in which a successful attack may lead to hazardous results.

Since the TSN standards are relatively new and prioritize the tight quality of service (QoS) requirements of critical systems, there is no comprehensive security solution against TSN-specific threats. Although a built-in traffic policing protocol is a part of the standards [4], it provides limited filtering capabilities to ensure that critical data streams receive sufficient resources. However, an effective solution requires monitoring time-sensitive streams over TSN protocols and recognizing malicious attempts. Accordingly, in this paper, we introduce an open-source network monitoring and intrusion detection system for IEEE 802.1 TSN protocols, `TSNZeek`. We extended an existing monitoring tool, Zeek (former Bro [5]), to analyze the new TSN protocols and detect several TSN-specific attacks. Zeek is a well-established open-source network monitoring solution that is popularly deployed in real systems as well as used in academia for research purposes[1]. We focus on the threats against two TSN protocols, IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) and IEEE 802.1Qcc Stream Reservation Protocol (SRP), since they (i) are the critical protocols addressing communication reliability and configuration, and (ii) have specific network behavior with new packets types and architectural aspects. Our contributions are listed as follows:

- We implement a new packet parser using a grammar definition language, *spicy*, to process SRP and FRER traffic via Zeek. To the best of our knowledge, this renders `TSNZeek` the first TSN-aware security monitoring tool.
- We implement an intrusion detection engine connected to Zeek to recognize several attacks described in our previous paper [1].
- We test our proposal in a TSN testbed and confirm that it successfully detects various threats with negligible overhead.
- We publish `TSNZeek` open-source together with the traffic and attack generation tools[2].

The remainder of this paper is organized as follows. Section II introduces the TSN protocols that `TSNZeek` is capable of processing, SRP and FRER. Section III presents the related work. Section IV describes the design and implementation of `TSNZeek`. Section V gives the experiment setup and evaluation. Lastly, Section VI concludes the paper.

II. Background on IEEE 802.1 TSN

This section describes two TSN protocols: IEEE 802.1Qcc SRP and IEEE 802.1CB FRER. In comparison to other TSN protocols, SRP and FRER introduce their own interfaces, packet structures, and configuration schemes that impose several security threats. Therefore, we mainly focus on them in this study.

A. IEEE 802.1Qcc Stream Reservation Protocol (SRP)

IEEE 802.1Qcc SRP introduces the resource reservation routines for time-sensitive streams to configure all TSN components in the systems satisfying tight QoS requirements. It proposes two main components: (i) a network configuration (CNC) entity to configure the TSN bridges remotely

[1]Zeek Project, https://zeek.org
[2]The source code is available at https://github.com/UHH-ISS/tsnzeek

ISBN 978-3-903176-57-7© 2023 IFIP

and (ii) a user configuration (CUC) entity to discover the endpoints [6]. It further offers three configuration schemes utilizing those entities.

- In the **fully centralized model**, endpoints directly communicate with CUC over a user/network interface (UNI) and request network resources for TSN streams with certain requirements such as the worst-case latency and inter-arrival times. CNC then configures the bridges according to the requests received by CUC.
- In the **centralized network/distributed user model**, edge TSN bridges, e.g., bridges that endpoints are directly attached to, forward SRP requests to CNC with network-wide visibility. Similar to the fully centralized model, it is responsible for configuring all TSN bridges.
- In the **distributed model**, TSN bridges forward SRP requests to each other to handle configurations individually.

SRP imposes a complex packet structure that allows an endpoint to specify various requirements via type-length-value (TLV) fields and recursive header groups. The respective standard [6] at Section 35.2 (p.105-134) explains the whole packet structure in detail.

B. IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER)

IEEE 802.1CB FRER enables redundancy against link failures by sending duplicate TSN flows, which are called member streams [7]. The talker sends member streams through multiple redundant paths configured in advance. An incremental sequence number is embedded in FRER frames within the R-TAG header, and the duplicate frames across the member streams have the same sequence number. The member streams rejoin at one or more points (e.g., at the listener or an edge bridge) in the network, where duplicate frames are discarded by their sequence number. Finally, the listener receives the original compound stream.

To discard the duplicate frames and obtain the original stream, FRER utilizes various stream recovery functions. These functions consider the sequence number of the most recently received frame to perform frame elimination. For instance, the match recovery function eliminates all the frames with a sequence number smaller than the recently observed one. The frame elimination helps to drop the duplicate packets received due to stuck senders or misrouting.

III. Related work

In this section, we briefly present related work that investigates the security threats against IEEE 802.1 TSN protocols and proposes security solutions.

Regarding security threats, in [8], the authors discuss the security threats in TSN-based industrial control systems. In [9], they analyze the impact of denial-of-service attacks on TSN protocols. In [1], we listed more than 30 attack vectors against several TSN mechanisms.

Some of the existing TSN protocols can be utilized against such security threats. For example, in [3], the authors employ IEEE 802.1Qav Credit-based Shaper (CBS) to prevent denial of service attacks. The authors of [10] and [2] combine IEEE 802.1Qci Per-Stream Filtering and Policing (PSFP) protocol [4] with a centralized controller to enforce ingress policies for packet inter-arrival times and rates, and stream bandwidth. Similarly, in [11], the effectiveness of PSFP is analyzed for the security of TSN-based automotive networks. Lastly, in [12], the authors discuss security policies via PSFP enforced by a centralized policy server.

There are more practical design and implementation efforts for the monitoring and protection of time-sensitive systems. In [13], the authors propose a monitoring system for the bridge and link status as well as time-synchronization accuracy, excluding security and intrusion detection aspects. [14] presents a security module to improve TSN protocols with hardware encryption and authentication. IEEE 802.1AE Media Access Control (MAC) standard also enables authentication, integrity, and confidentiality in Ethernet-based data traffic [15].

None of the works above offers security monitoring or an IDS for IEEE 802.1 TSN protocols with specific packet structures, traffic characteristics and requirements, and thus security threats. In contrast, we propose an open-source and extendable IDS solution to address TSN-specific attacks.

IV. TSNZeek: Design and Implementation

`TSNZeek` consists of monitoring and intrusion detection components shown in Fig. 1. The monitoring component processes and log the received TSN traffic. The intrusion detection component obtains the processed frames from the monitoring component and implements the attack recognition logic for TSN-specific attacks.

The overall operation of `TSNZeek` can be described as follows: the *event engine* distinguishes the incoming packets by their EtherType values, which is a standard header type of Ethernet frames. Then, *TSN parser* processes SRP and FRER packets according to the new parsing grammar that we developed. After parsing, the *broker* disseminates those frames to the *notice engine* and *detection engine*. While the former engine logs the traffic and specific events using built-in Zeek functionalities, and the latter implements our attack detection rules. The detection engine then pushes notices back to the notice engine when an attack is recognized.

In the rest of this section, we elaborate on those modules together with the notices and alerts that `TSNZeek` can raise.

A. Monitoring Component

The monitoring component corresponds to the original Zeek in terms of its working dynamics. It consists of the event engine, the TSN parser, the broker, and the notice engine. The TSN parser (blue, dashed lines in Fig.1) has been implemented from scratch. The event and notice engines (red, dashed, and dotted lines) are existing Zeek components that we extended further. For the engines, we used Zeek v4.2.0.

1) Event Engine: The event engine registers the protocol analyzers during initialization to specify the parsing grammar for respective protocols. Fig. 2 shows a sample registration process for SRP and FRER. When an SRP or FRER frame is

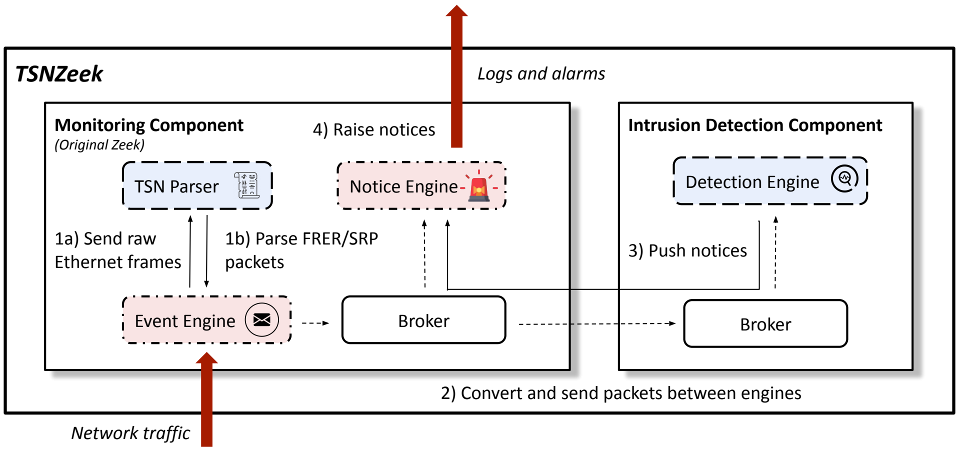

Fig. 1: The overview of `TSNZeek`. The blue/dashed blocks have been implemented from scratch. The red/dotted blocks are existing Zeek modules that we have extended and reconfigured.

recognized by its Ethernet header (e.g., with type `0x22EA` and `0xF1C1`, respectively), the event engine calls the respective packet parsing function implemented in the *TSN parser*.

```
event zeek_init() {
    if (!PacketAnalyzer::try_register_packet_analyzer_by_name("Ethernet",
        0xF1C1, "spicy::FRER"))
        print "Cannot register IEEE 802.1CB FRER analyzer";
    if (!PacketAnalyzer::try_register_packet_analyzer_by_name("Ethernet",
        0x22EA, "spicy::SRP"))
        print "Cannot register IEEE 802.1Qcc SRP analyzer"; }
```

Fig. 2: Registration of packet parsers.

The event engine registers further events to trigger logging facilities and inter-module packet dissemination via the broker. Both use the frame content provided by the protocol analyzers, i.e., for the received and parsed SRP and FRER frames. In the source code, all event registrations are implemented in the scripts *main.zeek* and *tsn.zeek* via Zeek scripting language.

2) TSN Parser: This module introduces the parsing functions for complex packet structures with many recursive header types. We implemented the parser using *spicy* v1.4.0, which is a grammar generation framework for network protocols and file formats[3]. We follow the packet definitions in the standards of SRP and FRER so that our parser can process any TSN traffic originated from a standard-compliant talker.

Fig. 3 shows an example *spicy* function to parse the talker information from an SRP frame. It extracts various traffic specifications and requirements given by a talker. Moreover, we define several FRER and SRP-specific header types, e.g., FRER R-TAG for sequence numbers, to be used within the parsing functions. In the source code, the files with *spicy* extension define the header types and parsing functions.

3) Broker: The broker is the built-in publish/subscribe messaging framework of Zeek. We implement three event topics in the broker: FRER, SRP talker, and SRP listener. Those topics are defined in *TSN.evt* in the source code. Whenever

[3]Zeek spicy, https://docs.zeek.org/projects/spicy

```
function makeTalker(obj: SRP::Talker): TSNZeek::Talker {
    local lendStationInterfaces = makeEndStationInterfaces(obj.
        endStationInterfaces);
    local ldataFrameSpecification = makeDataFrameSpecification(obj.
        dataFrameSpecification);
    local ltSpecTimeAware = makeTSpecTimeAware(obj.tSpecTimeAware);
    local luserToNetworkRequirements =
        makeUserToNetworkRequirements(obj.
        userToNetworkRequirements);
    local linterfaceCapabilities = makeInterfaceCapabilities(obj.
        interfaceCapabilities);
    return (
        makeStreamID(obj.streamID), makeStreamRank(obj.streamRank),
        lendStationInterfaces, ldataFrameSpecification,
        makeTrafficSpecification(obj.trafficSpecification),
        ltSpecTimeAware, luserToNetworkRequirements,
        linterfaceCapabilities); }
```

Fig. 3: Parsing function for SRP talker group.

a respective type of frame is received, the broker publishes its content, which is provided by the parser. The notice and detection engines subscribe to those topics and obtain the content of the frames for further analysis accordingly. For `TSNZeek`, we used Zeek Broker v2.2.0.

4) Notice Engine: The notice engine flags certain security events and logs received TSN traffic. We used the built-in notice facility of Zeek for this module. We configure the notice engine to log the received FRER and SRP frames partially to avoid an excessive amount of logs. A security event could be an anomaly in the configuration and network behavior, or a detected attack. The detection engine recognizes those events and then respective notice alerts are raised by the notice engine. The available notices are listed as follows.

- **N1.SRP. Excessive resource request:** If any talker demands more network resources than a predefined threshold, the notice engine raises this notice.
- **N2.SRP. Deviating resource request:** This notice alerts the resource demands that are marginally different from the previous SRP requests as it may indicate a malicious reservation.
- **N3.SRP. Too many requests:** It alerts if too many requests are received in a time interval, as it can indicate a stuck talker or an attack to exhaust the network resources.

- **N4.SRP. Changing existing allocation:** This notice alerts in case of an attempt to change an existing SRP reservation.
- **N5.SRP. Dangling resources:** This notice alerts the dangling resources if they are still not used after a predefined time after their registration, as it may indicate a faulty endpoint.
- **N6.FRER. Out of order frames:** An out of order FRER frame might indicate a malicious packet injection or a faulty endpoint. This notice alerts any out of order frame and if it should be dropped following the same mechanism as the stream recovery function in the corresponding TSN bridges.
- **N7.FRER. Excessive member streams:** FRER duplicates member streams based on the configured degree of redundancy. This notice is raised if the number of received duplicate packets for a stream is more than the degree of redundancy.
- **N8.FRER. Terminated member streams:** This notice alerts if a member stream is not active, as it may indicate a node or link failure as well as an attack.

B. Intrusion Detection Component

The intrusion detection component consists of the detection engine and another broker to communicate with the monitoring module. We implemented the detection engine purely in Python v3.9.2. It performs traffic analysis to (i) keep the current states of different streams and their configurations, (ii) make per-frame or periodical examinations to detect potential anomalies. Accordingly, it publishes the respective alerts via the broker to be logged by the notice engine. Note that while the first broker (attached to the monitoring component in Fig. 1) disseminates the frames from the event engine to others, this one establishes the communication between the notice and detection engines. It enables us to design the intrusion detection component as a standalone module that can be replaced by any other intrusion detection logic.

The detection engine in this component introduces a set of functions to detect various SRP and FRER threats listed in [1]. These functions are analogous to the rules in a rule-based IDS. Therefore, they are extendable to detect further threats simply as adding new rules. In the remaining of this section, we describe the detection functions together with the attacks they can recognize. We also note on the alternative placements of `TSNZeek` in the network, i.e., centralized, local, or peripheral in the network to detect the described attacks.

- **A1.SRP. Unusual SRP request:** An attacker can send malicious SRP requests to a CNC or an edge TSN bridge such as (a) demanding a bulky network bandwidth for a stream or (b) registering several streams to exhaust available resources. The detection engine detects such scenarios by comparing the requested stream traffic specifications extracted from the `TrafficSpecification` header of SRP frames with predefined threshold values for the maximum bandwidth and frame rates. It also keeps the rolling average of those values to recognize if an attacker requests stream reservations whose traffic characteristics significantly differ from the average.
- **A2.SRP. Flooding SRP requests:** An attacker can flood SRP requests to exhaust available resources quickly. The detection engine limits the rate of incoming requests and alerts for excessive requests. The rate limit is predefined and configured by the administrator.
- **A3.SRP. Changing existing allocation:** An attacker can forge an SRP request for an already registered stream to (i) reduce its reserved resources to degrade its service quality or (ii) increase its reserved resources to exhaust available resources without injecting any new stream that could be easily recognizable otherwise. The detection engine can automatically deduce if a request is accepted by checking the `TalkerStatus` group header of an SRP response. Then, for each SRP request, it checks if there already exists an accepted stream and alerts for one of the scenarios above.
- **A4.SRP. Dangling resources:** An attacker can reserve network resources to manipulate resource utilization without sending any real data traffic since it can also be detected and filtered by firewalls or network policies. For such cases, the detection engine periodically checks if the reserved resources are in-use. It alerts for the streams without any processed frames within a predefined time threshold.

`TSNZeek` should monitor all SRP traffic to detect the listed SRP-specific attacks. Therefore, while a centralized SRP configuration (see Section II-A) imposes a centralized `TSNZeek` deployment, a distributed one requires monitoring edge bridges.

- **A5.FRER. Forging fake sequence numbers:** If an attacker can observe the current sequence number of a FRER stream, it can inject malicious frames with that sequence number so that the legit frame would be dropped by the sequence recovery function in TSN bridges. If the attacker injects a frame with the upcoming sequence number, the detection engine alerts when it detects more than one frame with the same sequence number. Besides, it deduces the expected degree of redundancy, i.e., the number of expected duplicate frames for a stream, by processing the `NumSeamlessTrees` header in the `UserToNetworkRequirements` group header of the SRP request during registration of the respective stream. If the attacker searches for the legit sequence number by sending frames with the random sequence numbers, the detection engine raises an alert for an out of order frame.

The detection engine mimics the stream recovery function of FRER to keep track of the legitimate intervals of the expected sequence numbers. Thus, it needs to be configured with the same recovery function, match or vector recovery used by the TSN bridges in the system. This also requires monitoring TSN bridges locally to detect where exactly malicious frames are injected and dropped.

- **A6.FRER. Malicious rerouting:** If there are intersections between redundant paths, it eliminates duplicate packets being forwarded through the same bridge [16]. Instead of directly sabotaging the communication, an attacker could subtly reroute the redundant stream through intersecting paths to force FRER to drop the duplicate packets and hinder the redundancy. Therefore, `TSNZeek` examines the configured FRER routes, e.g., against malicious misroutings, intersecting paths etc. This requires the `TSNZeek` attached to the SRP controller,

TABLE I: The overview of the notices, attack detection functions, and other aspects of the intrusion detection component.

Protocol	Detection	Notices	Frequency		Deployment			Context	
			Per-frame	Period.	Central.	Local	Edge	Manual	Stateful
SRP	A1.SRP	N1.SRP, N2.SRP	✓	–	✓	–	✓	Resource threshold	Reservations
	A2.SRP	N1.SRP, N2.SRP, N3.SRP	✓	–	✓	–	✓	Rate-limit	–
	A3.SRP	N1.SRP, N2.SRP, N4.SRP	✓	–	✓	–	✓	–	Reservations
	A4.SRP	N5.SRP		✓	✓	✓	✓	–	Reservations
FRER	A5.FRER	N6.FRER, N7.FRER	✓	–	–	✓	–	–	Seq. numbers
	A6.FRER	N7.FRER	✓	–	–	✓	✓	–	Seq. numbers
	A7.FRER	N7.FRER, N8.FRER	✓	✓	–	–	✓	Timeout	Seq. numbers

i.e., centralized or hybrid, to access to the configuration of redundant paths.

- **A7.FRER. Triggering timeout:** An attacker can enforce FRER functions on TSN bridges to raise a `RECOVERY_TIMEOUT` event [7] if it can block all member streams of a FRER stream. Consequently, the expected sequence number of that stream is revoked. Once the attacker sends the first frame after this event, it becomes the valid originator of the stream with the forged initial sequence number. The detection engine recognizes the absence of the original member streams by measuring the time passed after the reception of the last frame of the respective streams. It also detects if the same stream has a new sequence number by per-frame examinations. Both can be detected by monitoring the edge bridge that the destination endpoint is attached to.

Table I summarizes attack detection and notices with several related aspects discussed above. These aspects are (i) how frequently `TSNZeek` investigates an attack, i.e., per-frame or periodically, (ii) how `TSNZeek` instances should be deployed to detect an attack, i.e., centralized, local, or to the edge, and (iii) what `TSNZeek` needs for the detection, i.e., manual configuration and the current state of stream reservations.

V. Evaluation

This section presents our experimental setup and evaluation results for the efficiency and performance of `TSNZeek`.

A. Experimental Setup

For our experiments, we set up a real TSN testbed shown in Fig. 4. It consists of three TSN bridges (TSN1, TSN2, and TSN3) connected in a ring topology. An endpoint (EP1) is attached to TSN3, and another (EP2) is attached to TSN2. `TSNZeek` is deployed on a computer with an Intel Core i3-9100 3.60Ghz CPU and connected to TSN1. A malicious endpoint (MEP) is also attached to TSN1 to conduct attacks. In our application scenario, EP1 first sends an SRP request for a resource reservation to stream a video. Then, it sends the data over two redundant paths, TSN2-TSN1-TSN3 and TSN2-TSN3, to EP2 and EP3 using FRER. `TSNZeek` monitors FRER frames forwarded over TSN1.

Since there is not any public TSN dataset including malicious traffic, we also implemented attacks described in Section IV-B in Python (available in the source code).

B. Results

We evaluated the resource usage and intrusion detection capabilities of `TSNZeek`. For resource usage, we measured the

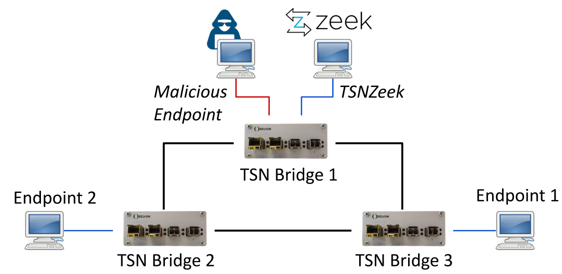

Fig. 4: Testbed setup.

CPU utilization of the monitoring and detection components, as well as the packet processing rate and delay of `TSNZeek`. For intrusion detection, the typical evaluation metrics for an IDS, e.g., accuracy, sensitivity [17], are not directly fitting for our rule-based IDS as its objective is detecting the specific threats according to the implemented rules. Therefore, we tested the effectiveness of the detection module against the attacks described in Section IV-B.

1) Resource usage: Since IEEE 802.1 TSN protocols define data link layer protocols, processing the events starting from such low-level communication may easily lead to high resource usage. Accordingly, we measured the CPU usage of `TSNZeek` for an increasing data load from 50 to 250 Mbit/s. This interval of data load is reasonable for the number of critical streams in a TSN network and mainly limited by the processing capacity of the TSN bridges in our testbed. We generated the load using *iperf*, which is a network speed test tool. Fig. 5 shows the resource usage of the monitoring component (Zeek module extended with TSN grammar, red and solid line), the detection component (Python program running at the control plane, blue and dashed line), and also the CPU consumption of native Zeek without TSN support (processing non-TSN Ethernet frames, black and dotted line).

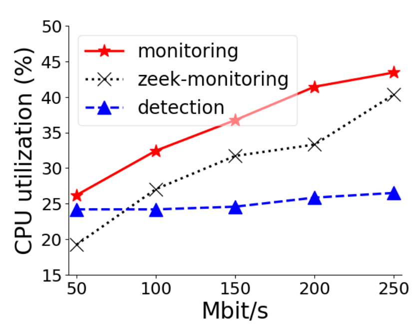

Fig. 5: CPU utilization of the monitoring and detection components.

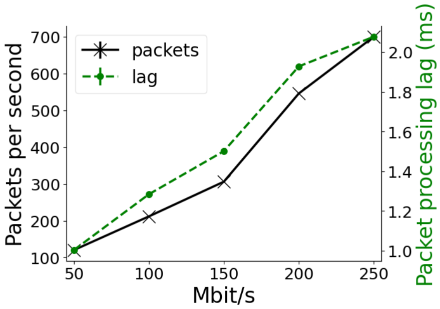

Fig. 6: Packet processing performance of the monitoring component.

When increasing the traffic load, the CPU utilization of the monitoring component increases from 25% at 50Mbit/s to 45% at 250Mbit/s. As shown in the figure, `TSNZeek` consumes only ∼5% more CPU power than the Zeek instance without TSN support. The detection module has a constant resource utilization of around 25% as it only processes singular events that are sent by the monitoring module.

Fig. 6 shows the packet processing rate and lag of `TSNZeek`. The packet processing lag describes the time passed between the reception and parsing of a frame. The figure shows that the packet processing rate (black, solid line) increases proportionally with an increasing data load without any packet drops, e.g., due to a potential congestion. Increasing load also leads to a higher packet processing lag of up to 2 ms (green, dashed line). For any lag in milliseconds, time-sensitive frames with submillisecond latency requirements may already be delivered before an intrusion alert. Although it is not critical for a monitoring module, a potential time-sensitive intrusion *prevention* system utilizing `TSNZeek` might require further improvements in data processing speed.

”**date**”: ”2022–10–26–16–41–02”,
”**timestamp**”: 1666802462.036670,
”**note**”: ”TSN::POTENTIAL_ATTACK_6”,
”**protocol**”: ”FRER”,
”**msg**”: ”Out of order frame is discarded for stream 29695 – seq.nr. 7148 < 54972”,
”**actions**”: ”Notice::ACTION_LOG”

Fig. 7: A sample intrusion alert in *json* format.

2) Intrusion detection: In our experiments, `TSNZeek` can successfully detect all the listed attacks in Section IV-B and raise the respective notices in real-time. Fig. 7 shows N6.FRER (in json format) in the log stream of the notice engine against the attack A5.FRER, i.e., the frame injection with the sequence number 7148, while the expected one is 54972.

However, we still observe redundant notifications in particular scenarios. For instance, when we connect an Ethernet hub between TSN1 and TSN3, e.g., extending the network with a non-TSN network component, a member stream delivers out of order packets due to the delayed frames. `TSNZeek` notices this as a malicious attempt because of highly deviating sequence numbers. Although this is unusual for strictly configured TSN systems, `TSNZeek` should still be configured considering such network conditions.

VI. Conclusion

Although IEEE 802.1 TSN standards propose emerging time-sensitive communication protocols for critical systems, they still lack security countermeasures against potential attack vectors. In this paper, we present the first open-source security monitoring and intrusion detection system, `TSNZeek`, for IEEE 802.1 TSN protocols. We implement `TSNZeek` by extending an existing monitoring tool, Zeek, with a new packet parsing grammar and the stream analysis engines addressing TSN-specific security events and attacks. We evaluate its resource usage and confirm that it can successfully detect various attacks against the prominent TSN protocols, SRP and FRER. For future work, we aim to improve our detection engine to minimize redundant and false alerts by accurately modeling the usual TSN behavior.

References

[1] D. Ergenç, C. Brülhart, J. Neumann, L. Krüger, and M. Fischer, “On the Security of IEEE 802.1 Time-Sensitive Networking,” in *IEEE Int. Conf. on Communications Workshops (ICC Workshops)*, 2021.

[2] P. Meyer, T. Häckel, S. Reider, F. Korf, and T. C. Schmidt, “Network Anomaly Detection in Cars: A Case for Time-Sensitive Stream Filtering and Policing,” *CoRR*, vol. abs/2112.11109, 2021.

[3] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, “DoS Protection through Credit Based Metering - Simulation Based Evaluation for Time-Sensitive Networking in Cars,” *CoRR*, vol. abs/1908.09646, 2019.

[4] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Bridges and Bridged Networks – Amendment 28: Per-Stream Filtering and Policing,” *IEEE Std 802.1Qci-2017*, 2017.

[5] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” in *7th USENIX Security Symposium*, Jan. 1998.

[6] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements,” *IEEE Std 802.1Qcc-2018*, 2018.

[7] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Frame Replication and Elimination for Reliability,” *IEEE 802.1CB-2017*, 2017.

[8] F. Fischer and D. Merli, “Security considerations for ieee 802.1 time-sensitive networking in converged industrial networks,” in *Int. Conf. on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pp. 1–7, 2022.

[9] M. Topsakal and S. Cevher, “Impact Analysis of Denial of Service Attacks in IEEE 802.1 Time Sensitive Networking,” in *30th Signal Processing and Communications Applications Conference (SIU)*, 2022.

[10] P. Meyer, T. Häckel, F. Korf, and T. C. Schmidt, “Network Anomaly Detection in Cars based on Time-Sensitive Ingress Control,” in *IEEE 92nd Vehicular Technology Conference*, 2020.

[11] F. Luo, B. Wang, Z. Fang, Z. Yang, Y. Jiang, and K. Demertzis, “Security Analysis of the TSN Backbone Arch. and Anomaly Det. System Design Based on IEEE 802.1Qci,” *Sec. and Comm. Networks*, 2021.

[12] R. Barton, M. Seewald, and J. Henry, “Management of IEEE 802.1Qci Security Policies for Time Sensitive Networks (TSN),” tech. rep., Technical Disclosure Commons, 10 2018.

[13] T. Bu, Y. Yang, X. Yang, W. Quan, and Z. Sun, “TSN-Insight: An Efficient Network Monitor for TSN Networks,” APNet, 2019.

[14] L. Muguira, J. Lázaro, S. Alonso, A. Astarloa, and M. Rodriguez, “Secure Critical Traffic of the Electric Sector over Time-Sensitive Networking,” in *IEEE 35th Conference on Design of Circuits and Integrated Systems (DCIS)*, 2020.

[15] IEEE 802.1 TSN Task Group, “IEEE Std. for Local and Metro. Area Net. – Media Access Control (MAC) Security,” *IEEE Std. 802.1AE-2018*, 2018.

[16] D. Ergenç and M. Fischer, “On the Reliability of IEEE 802.1CB FRER,” in *IEEE Int. Conf. on Computer Communications (INFOCOM)*, 2021.

[17] L. N. Tidjon, M. Frappier, and A. Mammar, “Intrusion Detection Systems: A Cross-Domain Overview,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3639–3681, 2019.