



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics

ICS Network Sensor

TDK

Semsei Dániel Andor (MSc Student)

Supervisor: Dr. Holczer Tamás

October 28, 2020

Contents

Absztrakt	4
Abstract	5
1 Introduction	6
2 Related work and technologies	7
2.1 Industrial Control System	7
2.1.1 Supervisory Control And Data Acquisition	7
2.1.2 Distributed Control System	9
2.1.3 Programmable Logic Controller	10
2.2 Attacks on ICSs	10
2.2.1 Stuxnet	11
2.2.2 Other examples	11
2.3 ICS security technologies	12
2.3.1 Firewall	12
2.3.2 Network segmentation and segregation	12
2.3.3 Authentication and authorization	12
2.3.4 Other technologies	13
2.4 Security Operations Center	13
2.4.1 SOC's goals	13
2.4.2 Sensor system	13
2.4.3 Security Information and Event Management	14
2.5 SOC solutions	14
2.5.1 Commercial products and solutions	14
2.5.2 Our solution	15
3 Development	16
3.1 The structure of the system	16
3.1.1 Operation of the sensor network system	17
3.2 Monitoring	18
3.2.1 Moloch	18
3.2.2 Generators	19
3.2.3 Generator class	19
3.2.4 Class loading	20
3.2.5 Generators and Splunk communication	20
3.3 Passive event generator	20
3.3.1 Operation of the generators	20
3.3.2 Scheduling of the generators	22
3.3.3 Test generators	23
3.4 Active event generator	24
3.4.1 Functioning of the generators	24
3.4.2 Scheduling and pumping	24
3.5 Algorithms used for scheduling	27
3.5.1 Testing of the algorithms	29

3.5.2	Test generators	32
3.5.3	Test of the algorithms	32
3.6	Splunk	35
3.6.1	Splunk vs Splunk free	36
3.6.2	Configuration of Splunk	37
3.6.3	Search and report	38
4	Evaluation of the system	40
4.1	Test scenarios	40
4.2	Scenario 1	40
4.2.1	Attack script	40
4.2.2	Generator scripts	40
4.2.3	Evaluation	41
4.3	Scenario 2	43
4.3.1	Attack script	43
4.3.2	Generator scripts	43
4.3.3	Evaluation	43
5	Conclusion	45
5.1	Future work	45
5.2	Result of the project	45

Absztrakt

Az ipari létesítmények kitűnő célpontok a kiber támadásoknak, mivel nagy fizikai és anyagi károkat lehet a szabotálásukkal okozni. Ezenkívül ezeknek a létesítményeknek a vezérlőrendszere elég bonyolultak, ezért elég nehéz követni, hogy éppen mi történik bennük. Ezek a rendszerek eredetileg el voltak szigetelve az Internettől, és ezért az operátorok nem nagyon törődtek a kiber biztonsággal. De napjainkra egyre nagyobb problémát jelent, hogy ezek a rendszerek egyre jobban ki vannak téve az Internetnek. Az ipari létesítmények vezérlését az Ipari Vezérlő Rendszerek végzik (Industrial Control System ICS).

Az ICS rendszerek elég komplexek, naponta rengeteg esemény keletkezik bennük, és habár ezek általában le vannak naplózva, hatalmas feladat lenne manuálisan átnézni őket, anomáliákat és gyanús tevékenységeket keresni bennük. A Biztonsági Művelet Központok (Security Operations Center SOC) erre lettek kitalálva. Ezeknek a rendszereknek az a feladatuk, hogy monitorozzák az ICS-eket, kövessék a részeit és a bennük zajló eseményeket. Ezekből meg hasznos jelentéseket és statisztikákat gyártsanak, hogy megkönnyítsék az operátorok munkáját. Ebben a projektben az volt a célom, hogy tervezek egy SOC keretrendszert ingyenes szoftverekkel egy ICS megfigyelésére, és bemutassam a SOC koncepció előnyeit.

Ahhoz hogy meg tudjuk védeni az ICS-t tudnunk kell, hogy éppen mi folyik benne, tehát adatokat kell gyűjtenünk az ICS-ről. Ehhez használtam aktív és passzív adatgyűjtési technikákat is. Az iparban a passzív módszer a preferált, mert nincs semmi ráhatása a rendszerre, csak lehallgatja a forgalmát, és aztán elemzi a forgalom csomagjait. Ez azért fontos, mert az ipari rendszerek ütemezése elég szigorú, a monitorozó rendszernek nem lehet nagy ráhatása az ICS-re, vagy különben a rendszer nem fog helyesen üzemelni. Habár az iparban nem ajánlott, használtam aktív technikákat is, ahol a monitorozó rendszer kérdegeti az ICS részeit időközönként. Egy speciális algoritmust használtam, amivel tudtam a ráhatást enyhíteni. Egyéb fontos tulajdonsága a monitorozó rendszernek a személyre szabhatósága, mert moduláris, tehát az operátorok létrehozhatják a saját moduljaikat, és bővíthetik vele az adatgyűjtő rendszert.

Miután gyűjtöttünk adatokat a rendszerről, fel kell dolgozni őket, hogy hasznos információkat szerezzünk. Ehhez a Biztonsági Információ és Esemény Kezelőre (Security Information and Event Management SIEM) van szükségünk. Ezeknek a szoftvereknek az a feladatuk, hogy összegyűjtsék az adatokat a monitorozó rendszerektől, rendezzék és tárolják őket, nyújtsanak keresési lehetőségeket. Ezenkívül riasztás és statisztika készítés is még egyéb hasznos funkciójuk, és ezzel csinálnak információt az adatból. A projektben egy piaci SIEM ingyenes verzióját használtam. Hiányoznak belőle bizonyos funkciók, de semmi olyan, amit ne tudtam volna megoldani. Miután konfiguráltam a SIEM-et is a keretrendszer technikailag készen állt a tesztelésre. Létrehoztam különböző forgatókönyveket és támadási szkripteket, amiket tudtam futtatni a rendszeren. Ezek a tipikus lépéseket tartalmazták, mint például felderítés, sérülékenység kihasználása, és a konkrét támadás. A végén a rendszer a lépéseket sikeresen detektálta, és továbbította a SIEM felé, amit le is tudtam kérdezni. Ennek a rendszernek a feladata a detekció, tehát amikor érzékelte a támadás lépéseit, akkor a célt sikeresen elértük.

Abstract

Industrial facilities are great targets for cyber attacks, because the attackers can cause enormous physical and economical damage by sabotaging them. These facilities also have complex control systems, that is why, it is hard to track, what is happening there. These systems used to be separated from the Internet, so the operators did not really bothered with cyber security, but nowadays it is a more relevant and serious problem, that these systems are exposed to the Internet more and more. These industrial facilities are controlled and managed by Industrial Control Systems (ICS).

ICSs are complex, many events occur daily in them, and maybe there are logs about these events. It is a tremendous task to manually analyse them, and look for anomalies and suspicious activities. That is why Security Operation Centers (SOC) were invented. These systems' purpose is monitoring the ICS, tracking the assets and events, creating useful reports and statistic for the operators, making their job easier. In this project my goal was to create a framework for a SOC with freeware software to monitor an ICS, and demonstrate the SOC concept's advantages.

To protect the ICS we need to know, what is happening there, so we need to gather information from the ICS. For this I utilized active and passive techniques to collect data. In the industry the passive methods are the preferred ones, because they have no impact on the system, they only listen the traffic of the ICS, and analyse its packets. It is important, because industrial systems' schedule is very strict, the monitoring system must have low impact on the ICS, or the system can malfunction. Although it is rather ignored in the industry, I used active techniques as well, where the monitor system is polling the ICS's parts regularly. I used a special algorithm, with which I could mitigate the impact on the monitored system. Other important feature is the customizable data collecting software, which is modular, so the operators can create and expand both passive and active monitoring system with their own modules.

After we gathered the data, we need to process it to have useful information, for this we need a Security Information and Event Management (SIEM) software. These software's task is collecting data from the sensors, indexing and storing them, providing searching option, triggering alerts, creating statistics from them. They technically create information from data. I used a commercial SIEM's free version. It lacked some useful function, but nothing I could miss or not get around. After the SIEM was configured the framework was technically complete, so I needed attacks to test the framework. I created attack scenarios and scripts, which could run an attack on the system. It had the typical steps like reconnaissance, exploiting vulnerabilities and the attack. But at the end all of the steps was detected by the monitor software and was forwarded to the SIEM, and could be queried by me. This system's purpose is detection, so as it detected all the attack steps, our goal was successfully accomplished.

1 Introduction

Nowadays more and more industrial facilities use automation technologies to make the operation more easy and effective. For these they are using Industrial Control Systems (ICS) to control and manage them. These facilities are great targets for cyber attacks, because the attackers can cause enormous physical and economical damage by sabotaging them. These facilities also have complex control systems, that is why, it is hard to track, what is happening there. These systems used to be separated from the Internet, so the operators did not really bothered with cyber security, but nowadays it is a more relevant and serious problem, that these systems are exposed to the Internet more and more.

As mentioned before these facilities are complex, so using some automated event collecting is advised, or else the security staff will waste their time to look through the log files looking for suspicious activities or security threats. That is why companies developed different security technologies through the years to make the security team's work easier thus rise the level of the security. Security Operation Center (SOC) is one of the various security technologies to solve this problem.

Collecting the security tools and asset in one place, where the security team can use them more conveniently is not really a new idea, companies can choose from huge variety of SOC products. But usually these commercial SOC systems are rather expensive. In this project I developed a framework for a SOC with similar functions like a commercial one. For this I used freeware software and my self written scripts. The goal is to prove the concept of that, we can create a SOC alike system with limited functions, even if we use freeware software. This system main function is collecting data from the protected system, then processing these data and providing useful information for the IT security staff. This is one of the major and vital function of a SOC, but usually they provide other functions, like incident response, vulnerability assessment, asset discovery/inventory to mention some of them.

The network sensor has two major part. The first part monitors the traffic and actively queries the protected system's thus gathering data from the system. The second part store and process these gathered data. It also provides user-interface, where the operators can analyse and examine them, create useful query from them. There are also some planned minor utility functions which make the data examine part easier, or make the data archiving possible.

In this document we will go through the parts of the network sensor system in details in the third section, after this we will evaluate the tests and end this entire project with the conclusion in the fourth and fifth section. But first we will discuss the related technologies and necessary definitions to this project in the second section.

2 Related work and technologies

Before we can speak about our system and its development, first we need to go through some definition and concept to understand what is the point of the network sensor system. In this section we will get to know the ICS world and its parts, discuss its attacks and defences, and get familiar with the SOC concept.

2.1 Industrial Control System

ICS stands for Industrial Control System, which purpose is as its name suggests to control some kind of industrial facilities, such as power plants, oil refineries, assembly factories. These systems main purpose is to automate, operate and monitor these critical infrastructures. ICS is a summary of technologies, systems and devices, they contain usually other subsystems like a SCADA subsystem, demilitarized zones, devices like PLCs and others. In this subsection we have a little overview about the ICSs and their aspects.

Before ICSs the industry relied on relays, mechanical solutions and radio systems, which were harder to learn, and did not provided clean interfaces, and easy-to-program possibilities to the operators. There were other difficulties, like different devices had very distinct interfaces, so training new staff take more time. After the appearances of the micro controllers, they started to spread in the industry as well. They provided more cleaner easier-to-use interfaces to the companies, although they still had vary interfaces, and used different protocols, so the compatibility still was an issue [1].

There are two types of ICS: Distributed Control System (DCS) and Supervisory Control And Data Acquisition (SCADA) systems, or their hybrid version. In the following sections we introduce these two systems.

2.1.1 Supervisory Control And Data Acquisition

SCADA stands for Supervisory Control And Data Acquisition. At first the SCADA system's primary goal was to provide an interface for the operators, who could monitor the facility via it. SCADA could be referred as the monitoring component, but usually it is referred as the entire control system. The other type of the ICS is the distributed control system, which will be discussed later.

SCADA systems usually have the following parts:

- Supervisory computer/Master terminal
- Remote terminals (PLC, RTU)
- Communication infrastructure
- HMI
- Historian

The next paragraphs explain these parts, and their roles in the system, and via them we could understand the operation of the SCADA systems [2].

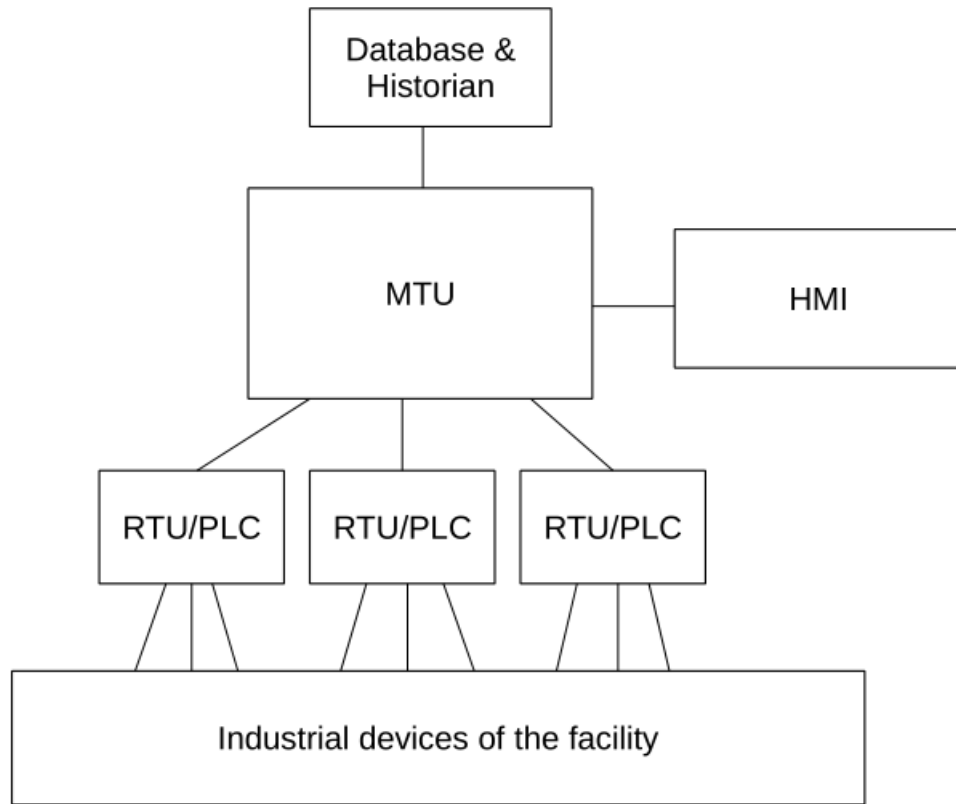


Figure 1: Structure of an average SCADA system

Supervisory computers Supervisory computers or master terminal units (MTU) or SCADA server. They have many names, many different implementation, but their role is the same, configure the PLCs and the RTUs, and gather data from them. On these machines the supervisory software are running, and usually the HMI software as well. As mentioned before, they supervise the controllers, and if some malfunction or failure has happened, they will notify the operators, who could handle these events. Also if the operators want to change the control loop, which are running on the controllers, they can do it via these servers. It is important, these servers cannot control or change the controllers in runtime. First the MTU has to stop the affected RTUs or PLCs, then the configuration could happen. That is why, they technically do not control the RTUs and PLCs, but configure them, so they can run automatically, and that is the point of the automation. Of course if something unexpected happens, the operators can intervene.

HMI HMI stands for Humane Machine Interface, and as its name suggests, it is a software, which provides a human-readable interface for the staff of the facility. They are usually running on the MTU, but running on a different terminal is possible as well. Via them the operators can reach the controllers, configure them, adjust their parameters, may set new control loops and read the data from them.

Control loop The program which is running on the controllers. They are cyclic and according to them the controllers are controlling the industrial devices. They can be written in many languages, for example structured text.

RTU Remote Terminal Unit is the full name, their role is very simple, they get a control loop from the MTU, and run it until some events occur. These control loop are written in LAD, which stands for ladder diagram. They are connected the devices and physical equipments of the facility, and by the control loop the controllers can operate them. From the sensors they gather data, which they forward to the MTU. If nothing unusual happens, they can work full automatically, no interventions needed.

PLC Programmable Logic Controllers, or the intelligent RTUs. They are more versatile, easier to configure and have more options in programming than the RTUs. They can process more difficult control loops, but their functionality is technically the same with the RTUs. There is more options in what language their control loop is written, but the most widely used are the ladder diagrams, function block diagrams and the structured text.

Communication infrastructure The infrastructure which connects the elements of the SCADA system, more precisely the MTU with the RTUs/PLCs, because it is common to have these in different places, even at long distances. They can contain security devices, like firewalls, networking devices, like switches, and many other utilities. These systems are using usually unique protocols for communication, which are usually made for the ICS/SCADA hardware and software in the system. They are very compact protocols, made by the vendor of the MTU, RTU and PLC devices.

Historian A database or service in the system, which can gather all of the data, what were proceed in the system. These data stored for decision support, later analysis, or for making statistic. This can be integrated in the MTU or the HMI.

2.1.2 Distributed Control System

Distributed control systems are used to control industrial facilities, usually located in one place. They are fully automated, so operator intervention rarely needed. They usually handle more control loop at the same time, and via them controlling the plants. There are many ways to implement a DCS, but each of them have one or two central controllers, which schedule and coordinate the remote controllers. The remote controllers are responsible for controlling the plant devices, such as sensors, motors, pumps, et cetera [3].

SCADA DCS comparison At first SCADA systems main purpose was to gather data from the facility and may let the operators intervene without the need of travelling to the facility, while DCS's was to control the process in a distributed manner. But nowadays they have many similarities like their main goal, which is to control the industry devices via control loop and remote controllers. From their former main purpose is why, DCS is process oriented, while SCADA is data-gathering oriented. Other difference is, DCS is process-state driven, which means, it is always scanning the outputs of the devices, and according to them will react, while SCADA is event driven, which means, it waits for specified events, and when they occurred, the operators can react. DCS operator stations are connected to the controllers directly, while SCADA operator station could be in a geographically different place from the plants. Although both have that advantage, if the connection is broken between the central control/master terminal and the remote controllers, the RTUs/PLCs still can operate with their control loop. But that is not unusual to have some DCS, and supervise them with a SCADA. Nowadays both have similar elements, and use some kind of remote controllers like PLC.

2.1.3 Programmable Logic Controller

Programmable Logic Controllers or PLCs are micro controllers. Their role is to run the control loop, thus automate the process, and control the devices of the facility. Control loops are programs, and they contain the logic, so when the controller get some data from its input from the devices, according to the logic and the inputs it will make decisions, and via its outputs will control the devices. PLCs also could use other various functions, like sequences, where they run and repeat a number of commands, without any logic, timers, which help them to synchronise the program, counters, which is another kind of logic and etc. Inputs and outputs are interfaces, via which the PLCs could communicate with the industrial devices of the facility. Inputs can come from sensors, while outputs are sent to actuators, servomotors, pumps, valves, etc. Inputs and outputs both could be analog and digital signals. Once the PLCs got their control loops from the master units, they can operate automatically without any operator intervention and aid. Of course if some event occurs, they may can handle, but at the more serious ones they may need the operators help.

There are two type of PLCs, single box/compact and modular/rack types. The single box ones usually have a CPU, memory, power supply, communication interfaces and inputs/outputs, they could be extended with additional I/Os, or other interfaces. The modular ones are built of modules, usually put in a rack, their units are usually in individual modules. They are more easier to expand than the compact ones [4].

2.2 Attacks on ICSs

At first the attackers were not really interested in attacking these systems, but nowadays more and more attacks are happen against ICSs. They are pretty good targets, because by this, the attackers could paralyse infrastructure of cities, sabotage serious projects, cause enormous damage to companies, corporations, countries. In this subsection we will examine some of more infamous attacks.

2.2.1 Stuxnet

[5] One of the most infamous malware against industrial control systems. It was a malware, which main target was the Iranian uranium enrichment facility, and sabotage its operation. It had three module: a worm, an LNK file and a rootkit. The worm's task was to spread the malware. When the worm infected a machine, it replicated itself, then looked for the next infection target. Also at every infection it checked, did it infect the target computer. One of the key feature was that, it could infect removable drivers, and via it infect the facility's system, because the target was not connected to the internet. Because of this the Stuxnet had to be very aggressive, infecting as many computer as possible to increase the chances to reach the target. Approximately 100.000 host was infected with Stuxnet, when it was discovered.

Step7 is a software to program Siemens PLCs. The Stuxnet's target was a computer with this software. When the Stuxnet infected the Step7's host, it replaced a dll, with which the software can reach the PLC, with its one, then renamed it, so furthermore the Step7 used the Stuxnet's dll file. Via this dll the Stuxnet infected the PLC, and loaded its code on the controller. It did not overwrite the original code of the PLC but extended it with its own malicious code. It sabotaged the centrifuges by changing their speeds. To keep unnoticeable the worm also sent forged data towards the operators, so they did not notice the unexpected behaviour of the centrifuges.

The LNK file's task was to automatically execute the worms copies, when they infected a host. And lastly the rootkit's task was to conceal the malware's existence, and the fact of the infection, on an infected host.

2.2.2 Other examples

Although Stuxnet is on of the most infamous attack on ICS, there are many other attacks, which case enormous damage to the victims, here is some example.

Black Energy Originally it was a DDoS malware, but later evolved, and got more other malicious function, like espionage functions. Presumably its third version was used to gather information about the Ukraine power plant, which led to another cyber attack against this facility in 2015, which caused power outages in Ukraine [6].

German steel mill cyberattack In 2014 an attacker first broke into the corporate network by spear phishing emails. At spear phishing the attackers send infected files concealed as normal ones to the target, and when the target open them, they infect the target's machine. After this the attacker could gain access to the industrial network, there he made some components to fail. This later caused physical damage in the facility [7].

Indian power plant breach In 2019 India's biggest nuclear power plant's defence was breached. The facility's internal network was separated from the outside, but still somehow it was attacked similar to the Stuxnet's case. The attack does not cause any critical damage, but showed the impotence of the air gap strategy. Presumably it was a targeted attack, and an unknown person connected the infected device to the internal network [8].

Cyberattack on HOYA In 2019 a Japanese optics manufacturer HOYA's production line was attacked. The attackers stole user credentials, because they wanted to use the system for cryptocurrency mining. The attackers were busted, because the operators detected the performance degradation of the servers. At the end the crypto mining plan was unsuccessful, the attack still decreased the facility production by 60% [9].

2.3 ICS security technologies

We discussed some attacks before, so next are the countermeasures, security technologies to protect our valuable target [10].

2.3.1 Firewall

Firewalls are software, hardware or systems, which can control the flow of the network traffic. They have rules, and according to them, they will handle the incoming packets. They are very customizable, which provides great flexibility. For example they could use whitelisting or blacklisting, the former rejects or drops every packet, which is not on its list, while the latter accepts every packet, which is not on its list. The type and variety of the firewall are enormous, and nowadays we cannot imagine any network with Internet connection without any firewall.

In ICS it is recommended to put firewalls between the internet gateway and the rest of the system. The ICS and the corporate network must be separated by firewalls as well. Other good use of the firewall is to partition the ICS into different segments with different set of rules, and put firewalls between them. The demilitarized zones are used as well to protect the ICS, DMZs use firewalls too.

2.3.2 Network segmentation and segregation

Network segmentation and segregation are strategies to separate the system into segments, and segregate them with firewalls. As before was mentioned, the segregation of the corporate and control network is one of these strategies. We could achieve better protection, if we make a DMZ, where we put those part of the control system, which are frequently communicating with the corporate network. These parts are usually the databases and historian, which gather and store data from the process. We could go further, if we segregate some part of the ICS as well. For example we could deploy firewalls between the supervisor and the remote controllers, or just set some rules in the controllers, like what they can accept.

2.3.3 Authentication and authorization

Authentication could be a good measure to protect part of the ICS from forged packets and commands. Usually the remote controllers are the targets of these attacks, when the attacker forges a command packet, and makes the controller believe, it is from the central control or the master terminal. It is very important to the remote controllers could authentic the incoming commands and accept them from the real master terminals only, or else the attacker could cause huge damage to the facility. With a good authentication and authorization algorithm we could make the attackers task much harder.

Sometimes the attack could come from inside by an employee, so authorization is important as well, in other words not let any employee to access anything in the facility.

2.3.4 Other technologies

There are other technologies and methods, with which we can achieve higher level of security, but they are not the typical solutions.

Honeypot Deception software, or hardware, with the purpose to fool the attacker, and make him believe he successfully attacked the real one. Honeypots could be classified in many aspect, have many functions, and usually have some kind of vulnerabilities, via which the attacker could attack it. After the "successful" attack, the honeypots usually monitor and log the actions of the attacker.

Unidirectional gateways Special gateways, which work only in one direction. For example we could put them between the historian and the corporate level, and we could ensure the data will flow only to the corporate level, so injections are not possible in theory.

Security Operations Center This technology will be discussed in details in the next subsection.

2.4 Security Operations Center

[11][12] Security Operations Center (SOC) is a place, where the security team can monitor the target system. It makes the IT security staff's job easier, by gathering every important security system and facility in one place. It strengthens the overall security of the facility. Also if a security incident occurred, the team can make the response from here. In this subsection we will go through the SOC concept and its point.

2.4.1 SOC's goals

Monitor, analyse, detect, respond are the main goals of the SOC in short. The security team needs to gather information about the protected system, so they need sensors in it. When they have data they need to analyse it so they can have real-time information about the system and its state. Then they need to have some technique, like anomaly or policy based detection, to detect the threatening activities in the system. When they are certain about something is happening, they can plan the response, and execute it. There are other tasks, like help the forensic investigation or create reports for the higher-ups.

2.4.2 Sensor system

Monitoring, gathering and storing the data is one of the main task of the SOC. For this we need a sensor system, which can passively or actively acquire data from the protected system. Industrial facilities have very strict schedules, we do not want to occupy them with security operations too much, that is why passive monitoring techniques are the preferred ones. These methods listen the traffic of the network, and acquire the data from it. While the active techniques interact with the other part of the ICS, and query information from them.

2.4.3 Security Information and Event Management

When we have data we need to process them, for this we have the Security Information and Event Management (SIEM) system. Its tasks are collecting data from the sensors, indexing and storing them, providing searching option, triggering alerts, creating statistics from them. It will create information from the data.

2.5 SOC solutions

SOC technology seems to be a profitable investment, so no wonder many big IT security company developed their own SOC products. In this subsection we will see some of these products, what functions and features they have. We can have a more clear idea about that, what features these SOC product should have.

2.5.1 Commercial products and solutions

Visibility One of the main feature is to provide visibility about the protected system. For this many product has asset discovery and asset inventory. Assets include every part of the ICS, like industrial devices, terminals, networking devices et cetera. It is good to know, what assets we have in the system. For this most of the SOC product has the asset discovery, which feature can automatically collect all the asset in the system. Then it stores their data in an inventory, where the operators can check on them. It makes possible for other features. These discoveries usually also provides topology and network maps, where the operators can see the system more clearly.

When we have an inventory we can make prevention steps as well. Vulnerability assessment and threat hunting other features, where we can get a more clear idea, what incidents and attacks could happen in our system, so we can prepare for them.

Asset tracking and integrity check is other security feature, where we can make sure nothing has modified or tampered our assets.

Detection Every SOC product has a sensor system, which is responsible for monitoring the protected system. When it detects suspicious activity or other policy breaking events, it could alert the security crew, or even make response automatically. The monitoring has two kind, we can monitor passively or actively. The passive monitoring is the preferred one, because we want as less impact on our protected system as possible. It is reflected on the commercial products as well, because all have passive methods but just a few mentions active techniques.

Passive monitoring means listening the traffic of the network, and according the detection method looking for threats and security events. There are three detection method used by most sensor system. Signature based detection is, where we have a database about the attacks signature, so the sensor system can look for these signatures. They can be various things, like byte order, or command order, or other characteristics. Anomaly detection is where we know the normal behaviour of our system, so the sensor system can look for differences from it. The last well known method is the policy based detection, where we create policies and rules, if they are broken, we can have our suspicion, something is happening. These policies have wide variety, like frequency of some packets, range of IP addresses, specific command presence, et cetera.

Data processing These SOC systems have wide variety of functions, which can process the data collected from the detection and visibility functions. Creating reports, triggering alerts, storing them for later forensic use, when an incident occurs. For this every SOC product has some integrated 3rd party systems like SIEMs. Of course there are usually more 3rd party system can be found for different tasks.

Other functions Asset inventory provide visibility, and monitoring provide detection, but there are other functions, which provide other security features. Authentication, authorization and privilege management is other feature, which every SOC system must provide. With them we can define, what user is able to do in the system.

As my internship I worked at an oil company who wanted to deploy a SOC system at their facilities. While I helped them to make decision, I met different solution from different vendors. At the end of the internship they have chosen 3 product, which have all of the features mentioned previously:

- Claroty platform [13]
- SCADAfence platform [14]
- Nozomi networks Guardian [15]

2.5.2 Our solution

In this project I developed a framework for a SOC, but because this is a thesis project I cannot have all the function and features like a real SOC product. In my project a focused on the detection part of the SOC system with a little bit data analysing as well. Our system utilizes the Moloch freeware software, which can capture traffic and visualise them as well. I wrote scripts, which can query the Moloch for packet captures, then investigate it and generate events from them. The scripts then send these events to the Splunk, which is a SIEM. Splunk can store the events, query them, and make statistic out of them.

To compare our system to a commercial one, we have the detection system, which can use anomaly or policy based detection, depends on how we create the rules. The Moloch also provides a communication partner map, which is not a network topology, but at least provides some visibility. Lastly we have a Splunk as our SIEM, where we use a free version, so it has constraints, but still it can provide us some statistic, and we also can query the events as well.

3 Development

In this section we will discuss the development process of the network sensor. First we will look at the entire system, then we will go through all of the parts one-by-one. We will see the techniques algorithms and technologies used to create this system.

3.1 The structure of the system

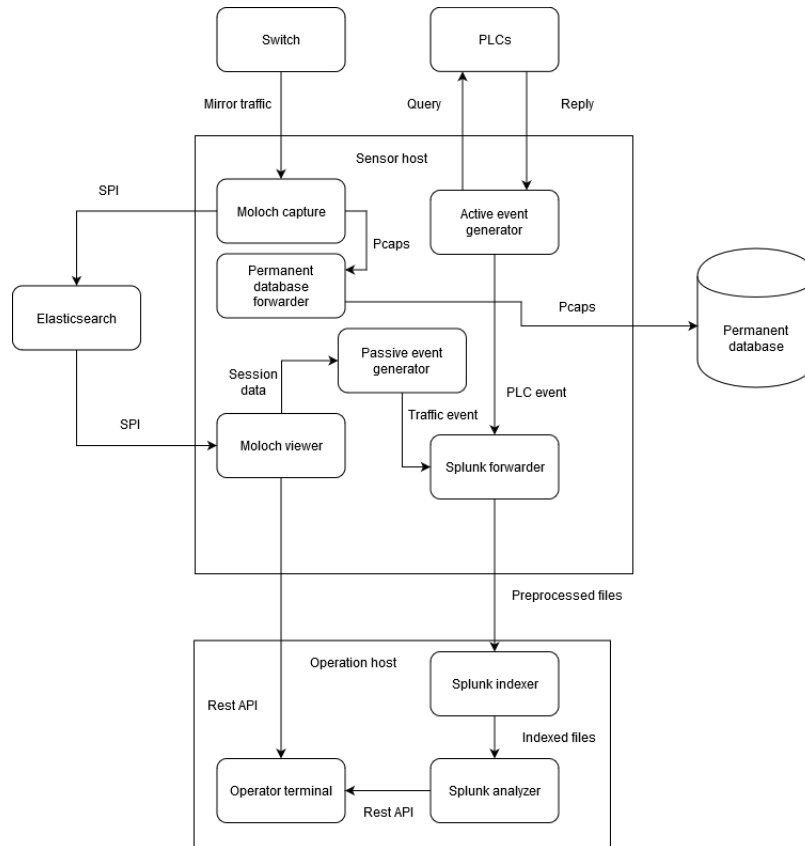


Figure 2: Structure of the system

The network sensor has 3 host to operate its tools and scripts. The first host has the packet capture freeware Moloch, the data preprocessing event generators, Splunk forwarder and the capture file archiving script. The second host has the Elasticsearch search-engine, which provides a fast querying of the capture files. The third host contains the Splunk, which is our SIEM system.

Moloch capture The capture service of Moloch. It listens an interface, where we must forward the traffic, and captures it, and stores it in PCAP files.

Moloch viewer It has two function, first it manages the capture files on the host, like deleting them, creating new PCAP file, et cetera. Second it provides a user interface, where we can query communication sessions, download PCAP files, examine the traffic. Also provides some statistic, like the number of the protocols usage. It also has an REST API, via which our scripts will be able to communicate with the viewer.

Passive event generator Data preprocessing script. This script queries the Moloch for capture files, inspects them, then generates events from them. Its generator set is easily extendable, so the operators can write their generators, thus deciding what the monitor system should look for. The events are stored in JSON, and will be forwarded by the Splunkforwarder to the Splunk.

Active event generator The other data preprocessing script. Similar to the active event generator, but instead of listening the traffic it queries the the part of the protected system like PLCs for data, and creates events from it. It is extendable, and stores the events in JSON format, which will be sent to the SIEM by the Splunk forwarder.

Splunk forwarder The before mentioned application, which monitors a directory, and when it found a new or a modified file in it, it will send the files content to the Splunk.

Permanent database forwarder Moloch viewer occasionally will delete PCAP files, when the hosts disk reaches a space usage limit, but it is possible we will need those files for a later forensic. For this we have a permanent database, which will store these PCAP files. The forwarders task is to upload these PCAP files to the database periodically.

Elastisearch Search engine which helps the Moloch indexing and querying its capture files' metadata. Elastisearch is memory intensive software while Moloch is disk intensive, so it is advised to be on a different host than the Moloch software.

Splunk indexer The indexer of the Splunk, its main task is managing the data in the Splunk.

Splunk analyzer Splunk analyser or search head. Its main function is sending search requests to the indexers, and gathering the replies from them. In our system there is only one indexer. It also provides a graphical user interface, where the operators can get more details and information about the data stored in the indexers.

Operator terminal It represents the host or terminal, where the operators can reach the different graphical user interfaces of the Moloch and the Splunk.

3.1.1 Operation of the sensor network system

First the network system has to collect data from the protected system. For this the protected system's whole traffic is mirrored and sent to the Moloch, which captures it. The Moloch indexes and manages this captured data's metadata with the Elasticsearch by sending the Session Profile Information (SPI) to the search Elasticsearch.

The passive event generator via the REST API queries the Moloch for packet captures, then generates event, then dump them in JSON format into a directory observed by the Splunk forwarder. The forwarder sends these preprocessed data to the Splunk indexer, which will index and store them for later uses.

The active event generator communicates with the devices of the protected system, and gathers data from them. It generates events from them, and via the Splunk forwarded passes the events to the Splunk indexer, which indexes and stores them.

The permanent database forwarder periodically queries the Moloch for capture files, then send them to a separated database.

The operator at the operator terminal via REST API can reach the Moloch and the Splunk web user interface, where they can examine the traffic and the events as well.

3.2 Monitoring

Monitoring when we observe the protected and gather information from it with different methods. In this subsection we will have a brief discussion of the monitoring and its structure, and the exact methods will be discussed in later subsections.

3.2.1 Moloch

Moloch is an open source program used for packet capturing, as well as the indexing and storage of those captured packets. It has a web interface that is intuitive and easy to use, and allows the browsing, searching, exporting and analysis of the captured packets. It stores the files in standard PCAP format. It can use REST API, so other programs can communicate with the Moloch.

One of the main tasks of our system was the observation of network traffic, we used Moloch to achieve this. Moloch needs a database in which to store the metadata belonging to the observed sessions. Moloch uses an engine called Elasticsearch to index these contents. This construct requires two separate hosts, one of which runs a Moloch instance, and the other an Elasticsearch instance. This is because Elasticsearch is quite memory intensive, and Moloch requires significant disk space. Moloch has two services that are required to run: molochcapture and molochviewer. While the former observes the network traffic, the latter does the administration and handles the utilities, such as:

- resource handling and allocation
- database handling
- etc.

3.2.2 Generators

Before we send anything to our SIEM, we need to preprocess the data. For this I wrote several scripts named event generators. These scripts main purpose is to collect information, generate events from the information, then pass the events to the Splunk forwarder, which will send them to the SIEM.

There are two kind of generator, passive and active. The passive generators query the Moloch for packet capture files, then investigate them, and according to what the script found, it creates JSON objects from them, and dump the objects into a directory observed by the Splunk forwarder. The active generators operate on the same principle, but instead of querying the Moloch for data, they query directly the ICS or networking devices for data.

To achieve customizability both generator method has its own main script, which at the start load the custom generator scripts, and manage them. These scripts will be discussed in details in later.

There are two thing, which is the exact at both passive and active generators:

- Generator classes handling
- The communication with the Splunk forwarder

These will be discussed in the following subsections.

3.2.3 Generator class

To achieve customizability we need some convenient and easy to understand method to add additional generators to the system, so the operators can write their own generator classes and add them without problems. The idea was to have a base class, and every other class must be inherited from that. I wrote the scripts in python, and python has inheritance, but does not have interfaces. So I had to use abstract class as my base class. When a operator wants to write a new generator script, they only has to inherit a new generator class from the base generator class and implement the necessary methods.

The first method has no parameters and returns with a string. This string is the name of the event. It can help to separate events generated by this generator from the other events. The second method is a valid Moloch filter expression. When the main script query the Moloch for PCAP files, it will use this expression to narrow down the result, so the generator has to work with less packet. The third method is where the operator has to implement the packet inspection part, because this method will get the list of packets as its parameter. This method must return with a list of dictionary objects. Dictionary objects in python technically JSON objects, because it contains key value pairs. Every event counts as a dictionary object, and the operator can decide what keys he will add to it. It will help later, when the events are in the Splunk, because we can search by those keys.

The previous class was the base of the passive event generator, the active one is different a little bit. It also has the event name method, but does not have the expression one. Instead it has a cycle time, because active event generators treated as tasks, which have cycle times. Cycle times are the time periods, in which the generator must run and finish once. This will be discussed in the active event generating subsection in details. The event generation method also different in the active generator's base class. Here it does not have any parameter, this method has to handle the querying. It must return with a list of dictionaries just like at the passive event generation. Here a dictionary equals one event as well.

3.2.4 Class loading

We have a base class and many inheritance of it. The main script needs to load these class in the run time, so it can use them. For this there is a method, where the main script need some information and can load these classes by itself. The loading of one class object:

```
generator_module = __import__(module_name)
generator_class = getattr(generator_module , class_name)
generator = generator_class()
```

Module name is the name of the python script without the .py extension, and class name is what the operator named the inherited class. With this method we can have a list, which contains the module and class name pairs of the generators, and load them in the run time. So when the operators completed their generator class, then they just have to add the module and class name to the class list, and the next run their class will be loaded and executed as well.

3.2.5 Generators and Splunk communication

When a generator class instance has run, and generated its events, it has to pass it to the Splunk forwarder. The Splunk forwarder is monitoring a directory, and when that directory has new content, the forwarder will acquire it and transmit to the Splunk. To make the data more structured I used the JSON format instead of plain string. That is why the generators' event generating methods must return with a list of dictionary, because they have the same struck with the JSON objects. The main scripts' task is to gather the dictionary lists and dump their content into JSON files. These files are named after the events, another use of the compulsory implemented event name method. When we examine the data in the Splunk web interface, we can sort the data by the source name, which are the JSON files, another reason why we should name differently our event generators.

3.3 Passive event generator

In the previous subsection we discussed the monitoring part of the network system as a whole, in this section we will get into the details of the passive event generation. The generator class loading and events handling over to the Splunk forwarder will not be discussed again.

3.3.1 Operation of the generators

Passive event generation is when the system monitor the traffic of the protected system and examine them. In industrial environment these methods are the preferred because they have no

impact on the industrial devices. There is two well known way to monitor the network. We can use middlebox devices called TAPs to put in our network, which will mirror the traffic going through them, and send to the sensor device. Switches usually has SPAN ports, which was created for monitoring purpose. Similar to the TAP devices, the switch will mirror its entire transit traffic and send to the sensor device via this SPAN port. These ports must have wide bandwidth, because they have more going through traffic than a normal port. In the network sensor, because it was created in a virtual network, the SPAN method was used. There is a switch, where every communication flow goes through, so its full traffic is mirrored and sent to the network sensor's sensor host. That host has an interface, which does not has IP address but only MAC address and monitored by the Moloch capture service. That is how Moloch captures the traffic.

As mentioned before the passive event generator has a main script, which first load the generator classes. Then it iterates through the generators, queries the Moloch viewer with the actual generator's expression for PCAP files. Next is parsing the PCAP into a list, which contains integer lists. The first list is the entire capture, while the integer lists represent the packets, and one integer in this list represents one of the byte of the packet. Next step is examining the capture according to the rule of the actual generator, and creating event from it. When the generator went through all the packet, it will return with list of dictionaries, which are the events. Before moving to the next generator the main script dumps the events into a log file, which is monitored by the Splunk forwarder, thus the events will be forwarded to the Splunk.

Querying Moloch To query the Moloch I used the "requests" python module. It has a function, with which we can send a GET request. It needs the previously created URL, and because the Moloch has authentication, it also needs an username password pair. It also has a session class, with which we can send all requests in one session. The reply will contain a PCAP file's bytes in Libpcap File Format.

Before the main script can send the request, first it needs to create an URL and specify some parameter. First it needs two time parameter, a stop time and a start time, which will define a time window. The Moloch is working with sessions, and when it receives a request with these two parameter, it will return those sessions packets, which session's last packet is in this time window defined by the two parameter. This interval is closed at the left end point and opened at the right.

To calculate these parameters the module needs a stop time date and a window length. It can acquire the time date in two way. The first is asking the system time from the operating system, the second is reading a previous one from a pickle file. The logic is the next, if the module finds the pickle file, it will read the stop time from it, else it will use the system time. After every generation cycle it serializes the successfully uploaded time frame's stop time into a pickle file, then sets the next time frame's start and stop time. The operators could easily reset the stop time by deleting the pickle file. The window length is a constant defined by the operators, and the start time is the difference of the stop time and the window length. Both stop and start time are Unix epoch time values.

The other parameter is the expression. Although it is optional, it can reduce the process time, because with this the Moloch could filter the packets, and send back only the relevant ones. Every generator class must have a function, which returns with a valid expression, or with a blank string.

Some useful expression example:

- protocols=[protocol name]: we could filter by the protocol, like tcp, udp, icmp etc.
- ip.dst ip.src: we could filter by the ip addresses
- port.dst port.src: we could filter by the port numbers

Of course there are numerous different expression. More information can be found at the help page of the Moloch webapi.

Parsing the reply The generator scripts cannot work with byte array, so it is the main script responsibility to parse the array into something readable. First I used scapy, but it had many disadvantages. Like the scapy's function needed a file, so I had to create a temporary pcap file, write the bytes into it, then give the file's name to the scapy function. The other problem is that, scapy is considered slow. Instead of scapy I created a function, which parses the reply's bytes.

The Libpcap File format starts with a global header, and it is followed by the packets. A packet has a header, and it is followed by the packet's bytes. The parser function has a "pointer", which at the start points at the first packet's header. It ignores the global header, because it does not contains any relevant information. Every packet header contains the count of the packet. After the packet length was read, it moves the pointer to the first byte of the actual packet. According to the packet length it puts the bytes into a list, of course after it parsed them to integer. After all of the bytes were put into the list, it appends the list into another list, which collects the packets. And finally it moves the pointer to the next packets header. At the end it returns with a list, which contains the packets as integer lists. Now the main script can pass these lists to the generator instances.

Generating events The last two step is generating events and handling them over to the Splunk. In our system we could speak about policy or anomaly based detection, they are pretty similar, because both looking for differences from the normal behaviour, but I would say it is rather policy based, because at the anomaly based detection there is some learning period, and here I just defined some rule to my generators.

After all generator inspected thoroughly their captures and generated the corresponding events, the main script with the "json" module's function will dump them in log files named after the events' name.

3.3.2 Scheduling of the generators

The event generating routine does not run all the time. It has periods when it is executed, then the main script waits for the period's end. To function properly it has to manage several variable, so no event is missed.

- Start time
- Stop time
- Time frame
- Moloch delay

The first three is needed to determine the time window for the query. The fourth is needed, because when the Moloch capture captures some packet, they will not appear immediately, Moloch needs some time to process it. So the generator is working with a little bit of delay.

There is two possibility, when the main script is started. It finds a pickle file or not. Pickle is a python module used for serializing data, in our case it will contain the last successful cycle's stop time. When there is a pickle file the main script knows, it needs to catch up with the event generation, so first it will run continuously without waiting. After it caught up, it will run its normal routine until it is stopped again. If there is no pickle file, it will use the current time minus the Moloch delay as its first cycle's stop time, and skip the catch up. It always calculate the start time by subtracting the stop time with the time frame.

3.3.3 Test generators

To test the passive event generator I wrote several test generator, which looked for simple events, like a ICMP echo request or specific Modbus requests. These test generators also a good example to test and demonstrate the extendability of the generator system.

ICMP event generator This is a pretty basic generator. It creates an event, when it detects an ICMP echo request.

When it gets the list of the packets, it iterates through it, and examine every packet. First it checks that, is it an IP packet by checking the value of the corresponding bytes. Here the 12th and 13th bytes contains the information in little endian format. If this is an IP packet, then it checks, is it an ICMP packet. If the 23rd byte's value is 1, then it is. Then it checks the 34th byte to acknowledge, what kind of ICMP packet it is. If there is value of 8, then it is an ICMP echo request, so the generator add an event to the event list. When it iterated through the packets it return with the event list.

There are other approach, which reduce the run time. We can set an expression, which will be used, when the main script queries the Moloch. If there we give an expression "protocols%3Dicmp", then the result of the query will, contain only ICMP packets. In this case, we do not need the IP and the ICMP check, therefore the script will be faster, because it will check less packet and only 1 check is required at every packet instead of 3.

Modbus event generator This generator is a little bit more complex than the previous, but still pretty simple. Like at the ICMP event generator, here the generator will iterate through the packets, and examines them one by one. It will create an event, when it detects some kind of writing request. Here the event will contain both IP address, the type of the write, and the addresses of the target of the write.

The generator first check, is it an IP packet, and then is the destination port is 502, which is the dedicated Modbus port. These checks can be replaced by setting expression, so the generator input packets surly will be Modbus packets. After this the generator will distinguish the write request into 4 group:

- write single coil
- write multiple coil
- write single register
- write multiple register

After it identified the type of the write, it appends the event with the address of the coil/register, and the IP addresses. If it is a multiple write it also include how many bit/byte were written, because address belongs to the first written coil/register. After it iterated through the packets, it return the event list.

3.4 Active event generator

We have already discussed the passive event generator so take a look at the other method the active event generator. This generator method is not really used in real life, because it has impact on the protected system, and we do not want that. In this subsection we will discuss the operating of this event generator and how I tried to mitigate its impact on the protected system.

3.4.1 Functioning of the generators

At the active event generation our generators are querying the control devices like the PLCs, or network devices like the switches. For this usually they need modules to have the function to communicate with them. In our case the generators used the Snap7 module to be able to communicate with Siemens PLCs, which are using the S7comm protocols.

The active event generator has a main script as well. It has similar task just like at the passive event generating. It loads the generator scripts, it runs them, then gathers the events from them and handles over the Splunk forwarder. But there are major differences as well. First it does not have to query anything, it is the generator instances' task. But it has to schedule them. Scheduling has two part: creating the schedule, then calling the generators according to the schedule.

3.4.2 Scheduling and pumping

At active event we can look the generators as periodic tasks. These generators will run in a real-time environment where every second counts, and querying the PLCs take time and resources from the PLCs and the network, like CPU time, memory space, bandwidth of the network, et cetera. So we do not want to run these generators sequentially quasi at the same time. This would happen, if we scheduled them simply without any additional technique. Also we have to pay attention to have every task run and terminate in their period time. So we want to schedule them as evenly as possible, avoiding sequential running and burst of data in the network. How I schedule these tasks approximately evenly, and what algorithms and techniques I used to achieve this, that will be discussed more deeply in a later subsection.

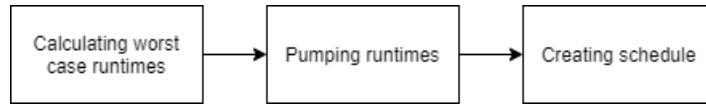


Figure 3: Steps of the scheduler

After we have the list of the generators, we need to find out their runtime, so first the script iterates through the list, and calls their event generation method, then writes the result into a log file. It will take a note of the duration of the time, and if it is longer than 500 ms, it will be the runtime, else it will be 500 ms. Then it creates a list of Task objects. A Task has runtime, cycle time and the generator, which has to run in every cycle. Then the pumping process is following.

The pumping technique is, with which we want to achieve evenly distributed run of the generators. This will be discussed in details in a later section, but in brief we increase the runtime of the task as much as possible with some idle time, and with this we create gaps between the tasks, so when they run, there will be always at least one gap between two task, so they will not run sequentially.

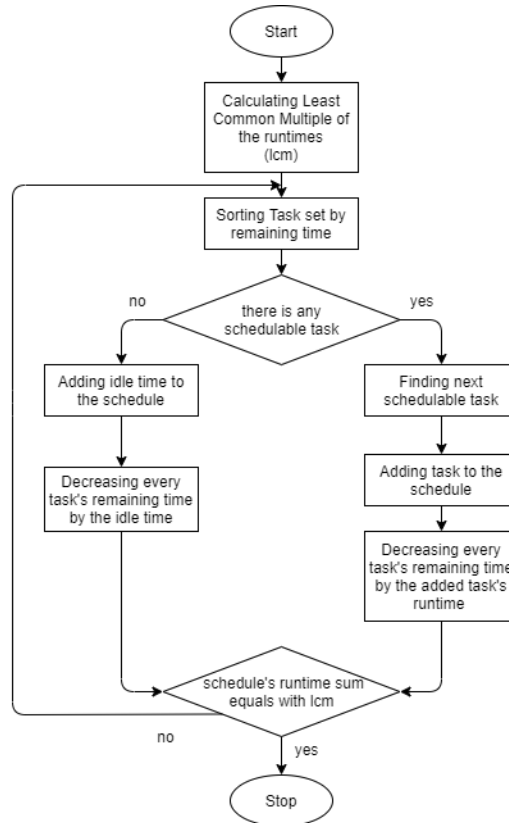


Figure 4: Scheduling process

After the task's runtimes were pumped, we can hand the new task list to the scheduler, which will create the schedule. The scheduler uses the earliest deadline first algorithm, where the task, which is closest its deadline will be scheduled. Every Task object stores its cycle time and the remaining time till its deadline. At the start these two value are equals. When the algorithm starts, the scheduler sorts the task list by the remaining time in increasing order. Then it iterates through the list, until it finds an executable Task. It schedules the task, which means, it appends a new element to the schedule, where this element contains the unpumped runtime and the instance of the generator belonging to the Task. Every Task object contains the original unpumped runtime and the pumped runtime, so the scheduler can append another element to the schedule, which contains the difference of the original and the pumped runtime. This element also contains a None object instead of a generator object, so the main script will know, this time it does not have to run any generator. After the schedule got its new element the scheduler first change the Task's pumped runtime to negative, then the scheduler decreases every Task's remaining time by the amount of the scheduled Task's pumped runtime. The negative runtime indicates that, the Task was scheduled. After the remaining time reduction, the scheduler checks, is there any Task with negative or zero reaming time, which means their cycle has ended. If the scheduler finds any of these, it will set its pumped runtime to positive and sets its remaining time to the proper value, which is the sum of the Task's cycle time and previous remaining time. If the scheduler cannot schedule any task, it will schedule an idle time, where the runtime is the smallest remaining time of the Tasks, and the generator is None.

Listing 1: The implementation of scheduling

```
def create_schedule(task_set):
    schedule = []
    lcm = get_lcm(task_set)
    while lcm > 0:
        task_set.sort(key=lambda x: x.t, reverse=False)
        for next_task in task_set:
            if next_task.c > 0:
                schedule.append((next_task.c_real, next_task.generator))
                schedule.append((next_task.c - next_task.c_real, None))
                next_task.c *= -1
                for task in task_set:
                    task.t_remaining += next_task.c
                    if task.t_remaining <= 0:
                        task.c *= -1
                        task.t_remaining = task.t + task.t_remaining
                lcm += next_task.c
            break
    else:
        idle_time = task_set[0].t_remaining
        schedule.append((idle_time, None))
        for task in task_set:
            task.t_remaining -= idle_time
            if task.t_remaining <= 0:
                task.c *= -1
```

```

        task.t_remaining = task.t + task.t_remaining
    lcm -= idle_time
    return schedule

```

The schedule length is equals with the least common multiple of the Task's cycle time, so every Task's periods will end at the end of the schedule, thus the main script can repeat the schedule. When the schedule is done, we can move to the last phase of the active event generation.

After we have the schedule, the last thing is to execute the generators according to it. Every element of the schedule is a tuple with an integer and a generator object. The script takes the next element, and if the generator is not a None object, it calls the generator's `generate_events()` method, which will return with a dictionary list, which is the events. Then the script take the events, and dumps them into a log file named after the result of the generator's `get_event_name()` method. If the generator in the tuple is a None object, the script will sleep for the amount of the integer in the tuple. When the script ends with the schedule it will repeat it till the program is terminated.

3.5 Algorithms used for scheduling

Basic definitions First we will talk about the periodic tasks, the schedulability tests and the cooperative Earliest Deadline First (EDF) scheduling algorithm.

Periodic tasks have 4 attribute: release time, computation time, period time and deadline. Release time is the first time when a task can run, in our case this will be $t=0$, so every task can run and be scheduled at the start. This will be ignored further. The computation time or worst case execution time is the time, which the task takes to run and terminate, it will be indicated as C . The period time is the interval, in which the task must run one time, it will be indicated as T . The deadline is the time, until when the task must terminate, in our cases the deadline will be equal with the period time.

Now we have a task set, which contains n amount task, where we know how many time take them to run, and how long their period times. The next question is, are they schedulable, or if we give them to a scheduler, can it schedule them, so none of them miss their deadline, and run once in every period. For this we have the schedulability tests, more precisely there are two [16].

$$\sum_{i=1}^n C_i/T_i \leq 1 \tag{1}$$

The first schedulability test is the utilization test. This will test, if the CPU is not overutilized. Of course in our case there will not be CPU but network. The quotient represents the utilization of the CPU. If the computation times and period times divisions quotient's sum is bigger than 1, then some tasks surely will miss the deadline, because there will not be enough time to run every task in their period time.

$$1 < i \leq n \quad T_1 < t < T_i \quad t = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{t-1}{T_j} \right\rfloor * C_j \tag{2}$$

The second test is the demand test. For pre-emptive EDF the first utilization would be enough, but unfortunately our scheduler algorithm must be cooperative, because we cannot interrupt or put on hold a message transmission on the network. For this we have the demand test. For the test the tasks must be sorted in a increasing order by the period time. In this test we check: is there enough time to run every higher priority tasks, and still left enough time to run the i -th task. We check it in every t until we reach the end of the period of the task And we check it for every task.

Now we can test if a task set is schedulable, next is the algorithm to schedule them, and I chose the earliest deadline first or EDF algorithm. There are other cooperative scheduler options like the first-come first-served (FCFS), or the shortest job first (SJF) as well. The FCFS is not a great option for periodic tasks, because there are no priorities, so a task with longer computation time could make another task miss its deadline, even every task has the same computation time. The SJF is not fitting because there the priorities are specified by the computation time, and in our case every task has the same computation time hence they would have the same priority. There are still the rate-monotonic algorithm, but it has no cooperative variant, and cannot achieve as much CPU utilization as the EDF can [17].

With EDF algorithm the scheduler checks, which task is the closest to its deadline, and schedule it. In pre-emptive it checks every time, is there any task closer its deadline, in cooperative, it only checks, when the actual running task is terminated.

Now we have schedulability test and scheduling algorithm, but if we use them simply, we will get a schedule, where every task will run sequentially, then wait for the ending of their period time. It will make them run in a burst manner. We need another techniques to schedule them, so they will run more evenly, and will not occupy the PLC continuously for a longer time. These techniques will be discussed in the next section.

Algorithms I had two idea how to achieve even running of the task, and the first was to add a dummy task, which will get between the tasks, so they will not run continuously. The goal is to have a dummy task with small period time, and high computation time. Small T required, so the scheduler will schedule it often, while high C means, longer time while it does not let other task to run. First the algorithm defines the values of the dummy task. The dummy's T is one time unit lesser then the shortest period time prior the dummy. The dummy's initial C is one unit lesser then the dummy's T . Then the algorithm adds it to the task set, and checks the set with the schedulability test. if it is not schedulable, then it decreases the dummy's C with 1, then repeats the test. If the set passes the schedulability test, then you can create a schedule with the EDF scheduler. If the trying reaches where C would be 0, then unfortunately the set cannot be scheduled with the dummy. You also can start a test without the dummy, just to make sure the basic set is schedulable.

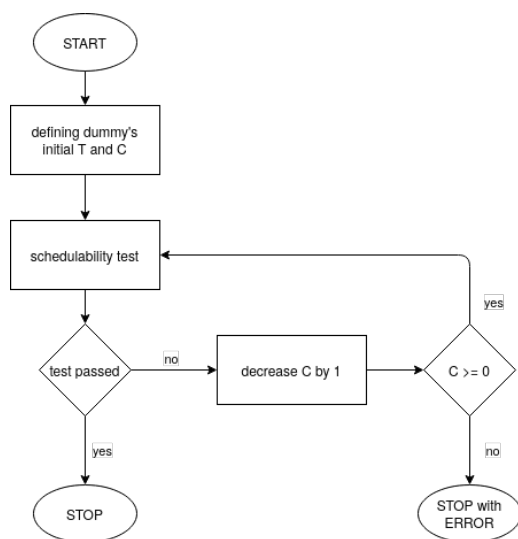


Figure 5: Steps of the dummy technique

The second idea was to pump the tasks computation time, till they are still schedulable. In our case every tasks C are very low, so we can simply consider all of them as one time unit. Then we can start the pumping process, which is increase every C by one unit, then make a schedulability test. Repeat this process until the set is still schedulable. We can make a schedulability test with the basic set as well.

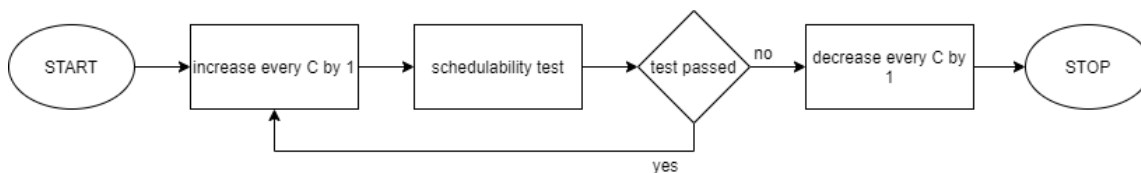


Figure 6: Steps of the pumping technique

3.5.1 Testing of the algorithms

We have now 2 technique create, and stretch out the time gaps between the task, so in this section I will test them, and evaluate the result, and decide which one to use. The next task set will be used for testing: (1, 30), (1, 40), (1, 70), where the first number is the computation time, while the second number is the period time. This is a relatively small task set, but it demonstrates pretty well the differences between the technics or their absent. The tests will calculate the burst lengths, or the number of tasks running sequentially, and the gap lengths, or how many idle time unit is between two running task, when nothing is running.

First I schedule the task with EDF without any additional technique, this test will give us the control result, so we can compare the other results to this.

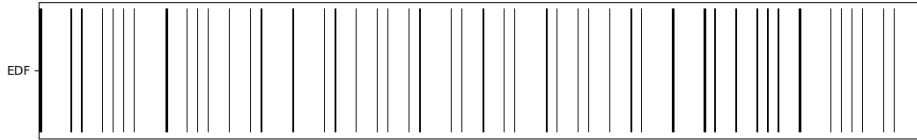


Figure 7: EDF without any technique

Smallest gap	0
Largest gap size	29
Average gap	12.5
Smallest burst	1
Largest burst	3
Average burst	1.2708

Table 1: EDF without any technique: statistic

As we can see There are many black line, some of them thicker than the others, those are the burst, where more tasks run after each other. The smallest gap size and the smallest burst size indicate, there is sequentially running task, which we want to avoid. The next test is task set with dummy task.

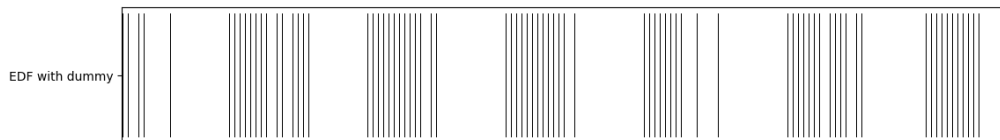


Figure 8: EDF with dummy

Smallest gap	0
Largest gap size	55
Average gap	12.7541
Smallest burst	1
Largest burst	3
Average burst	2.0498

Table 2: EDF with dummy: statistic

It seems worse than without dummy. There are still burst, the average burst size increased, which means there are more longer burst. We can see on the plot there are intervals where the tasks seem to be scheduled evenly, but there are huge gaps between these intervals. The next test is task set with pumped computation times.

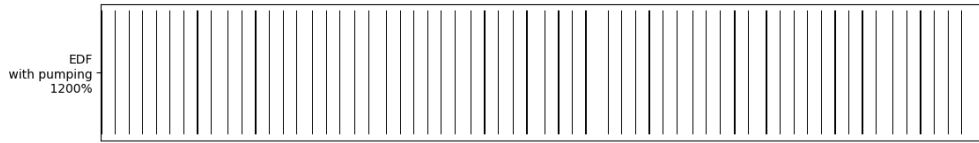


Figure 9: EDF with pumping

Smallest gap	12
Largest gap size	20
Average gap	12.5833
Smallest burst	1
Largest burst	1
Average burst	1

Table 3: EDF with pumping: statistic

It seems much better. First there are no sequentially running task, because the smallest gap is larger than zero. Then the arrangement of the black lines on the plot seems more evenly. I made some additional test with pumping, where I did not pumped fully the C values of the tasks.

+3 pumped	
Smallest gap	3
Largest gap size	29
Average gap	12.5
+6 pumped	
Smallest gap	6
Largest gap size	29
Average gap	12.5
+12 fully pumped	
Smallest gap	12
Largest gap size	20
Average gap	12.58333

Table 4: Difference between different amount of pumping

As we can see, with every additional pumping the difference between the smallest and the biggest gap are decreasing, and the average gap size are increasing. In the most even case, the smallest

and the largest gap would be equal, so it indicates the improvement. Because of the nature of the pumping technique, every gap at least as big as the pumped C minus 1, so the increasing average means there are more larger gap than the previous iteration, so more gap approaches the optimal gap size.

From the testing we can see the dummy technique was a failure, while the pumping one an improvement. The dummy ones biggest fault, is when the dummy task has run, it will wait for the ending of its period time, so the scheduler will schedule the other task, and because the computation times are really small and the period times are large, it can schedule many task, before it could schedule the dummy again. While the pumping technique provides a minimal gap between every task, so there will not be any burst ever. The decision seems trivial and so I chose the pumping technique for scheduling the generators at the active event generation.

3.5.2 Test generators

I implemented 2 kind of generator for testing. The first kind is using MODBUS for communicating with a Schneider Modicon M241 PLC, the other is using S7COMM protocol for communicating with a Siemens LOGO!8 PLC. The exact functions of these generators are not important, the goal is to have some traffic on the network. For the MODBUS one I used the pymodbus modul [18]. At this generator it creates a MODBUS client first, then communicate with the PLC via this. It write the PLC coils, then read its coils and holding register. At the S7COMM one I used the snap7 module [19]. First it creates a client, then writes some byte into the memory of the PLC, then reads some bytes from the memory. These two is the base, and every generator in the test are the exact copy of them except their cycle time and event name.

3.5.3 Test of the algorithms

I created 6 cases, a general one, a dense one and a rare one, all of them with and without pumping variants. A general one contains 3 generator with 10 second cycle time and 2 generator with 30 second cycle time. These cases represent a general situation, where we have frequent and less frequent generators. The dense one contains 8 generator with 5 second cycle time and 2 generator with 12 second cycle time. These cases represents a situation where the network is used frequently. The rare one contains 5 generator with 30 second cycle time. These cases represent a situation, where are no many generators, it will rather show us the effect of the pumping technique.

X	T = 5 sec	T = 10 sec	T = 12 sec	T = 30 sec
General approach	-	3	-	2
Dense approach	8	-	2	-
Rare approach	-	-	-	5

Table 5: Generator counts of the different approaches

Test results and evaluation At the test we will compare the pumping and no pumping variants. There will be diagrams where the the x axis will represent the time with 0.5 ms/unit steps, while the y axis will represent the sent bytes.

General test cases

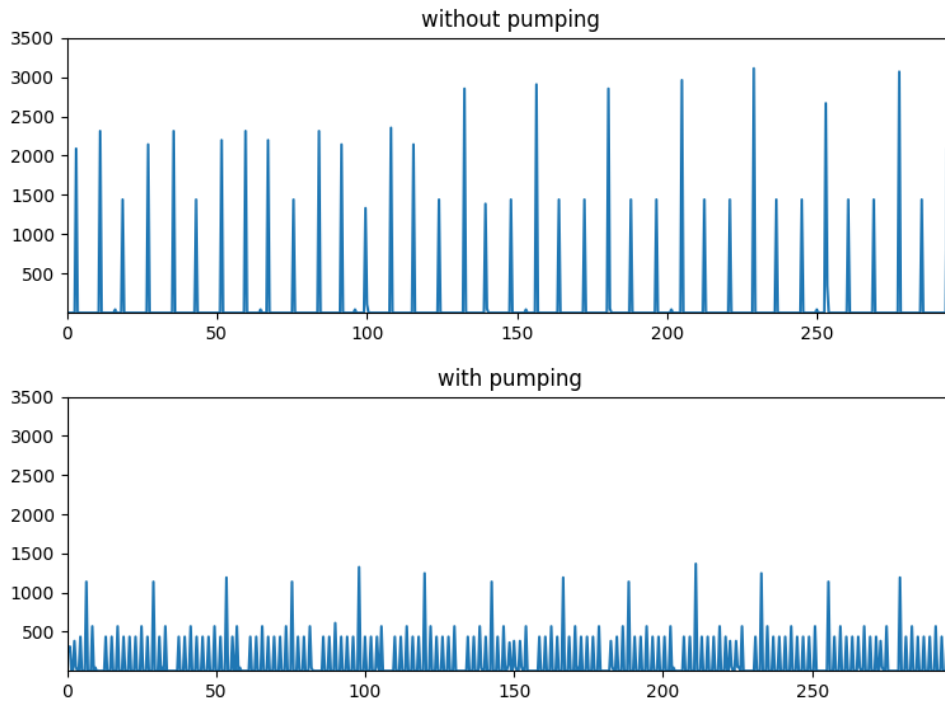


Figure 10: General approach result

We can see on the diagram, when pumping was used, there are less and smaller spikes, and probably the higher ones, where a the more byte heavy generator was used.

X	no pumping	pumping
smallest gap size	0	2000
largest gap size	8500	4500
average of the gaps	2227.27	2227.27

Table 6: General approach statistic

The smallest and largest gap difference has get smaller as well when pumping was used.

Dense test cases

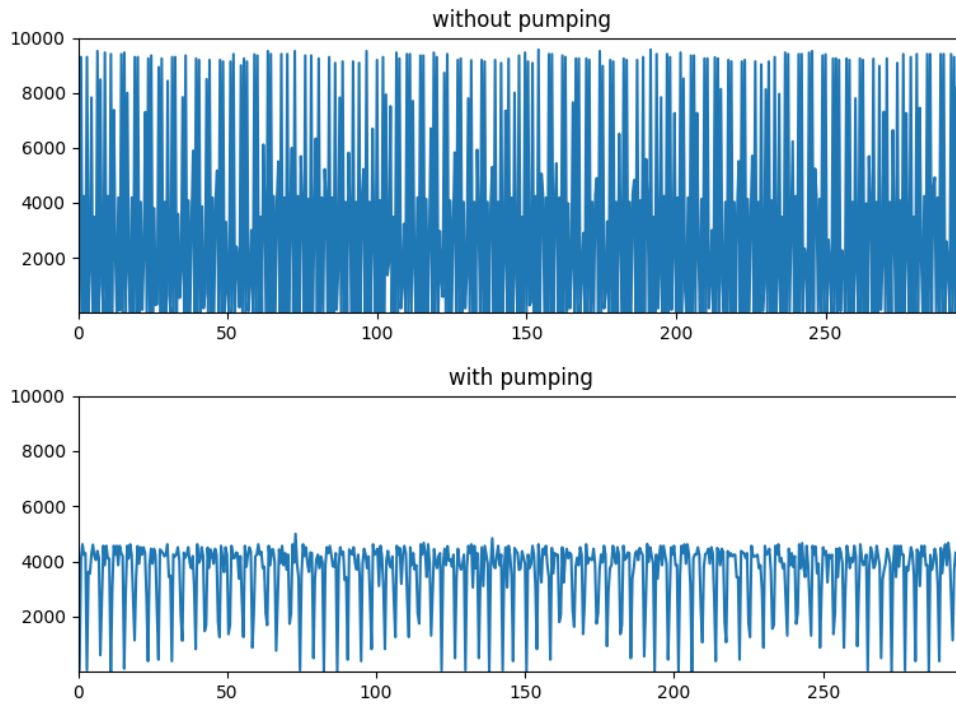


Figure 11: Dense approach result

We get similar result like in the previous, there are larger spikes when pumping was not used.

X	no pumping	pumping
smallest gap size	0	55
largest gap size	1000	615
average of the gaps	66.04	66.04

Table 7: Dense approach result statistic

In these cases the script cannot pump that amount like in the previous cases, but we still can see some improvement, the difference of the smallest and largest gap got smaller with pumping.

Rare test cases

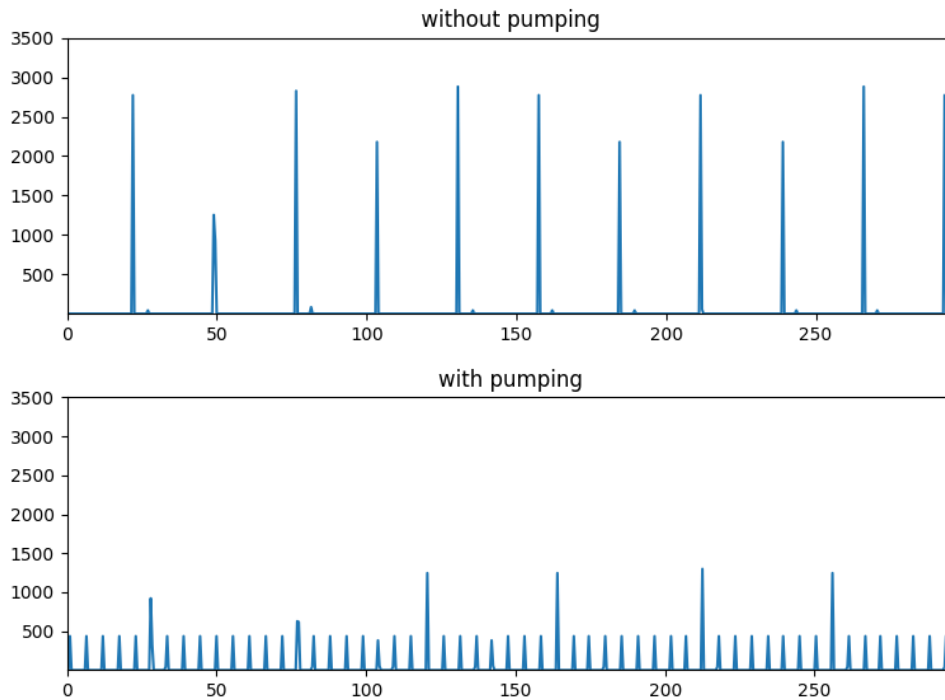


Figure 12: Rare approach result

The result is similar like in the previous test cases, with pumping we got smaller spikes. Here we can see the network usage are distributed more evenly.

X	no pumping	pumping
smallest gap size	0	5499
largest gap size	27500	5504
average of the gaps	5500.0	5500.0

Table 8: Rare approach statistic

These test cases represents the best of the effect of the pumping, because here the script could pump the most.

3.6 Splunk

[20] Splunk is a SIEM (Security Information and Event Management) system, which allows us to analyze large amounts of data, and correlate events generated from this data in real time. It has three components: Universal Forwarder, Indexer, and Splunk Web/Analyser.

The Forwarder is on the sensor host, and is responsible for the forwarding of the data to the Indexer. It is not particularly heavy on any resource, as its only task is the forwarding of data. Therefore, multiple Forwarders scale well together, multiple could be used within the same host, which would allow us several parallel data streams, which could all be targeted towards the same Indexer. In our system there is only one Splunk forwarder.

The indexer creates events from the incoming raw data, and stores them in indices. Those indices called buckets, and they are ordered by time. The indexing is happening in a pipeline manner, the first step is analysing the raw data (for example: acquiring the timestamps and other useful information from the data). The next step is arranging them in indices. Furthermore the indexer other task is to handle the search requests. The Search Head is responsible for managing the searches. For example: if there are more indexer, the Head will gather the results, and consolidate the information.

A single search head performs the search management function. It handles search requests from users and distributes the requests across the set of indexers, which perform the actual searches on their local data. The search head then consolidates the results from all the indexers and serves them to the users. The search head provides the user with various tools, such as dashboards, to assist the search experience.

Splunk Web is a graphical user interface, which is considered as the formerly mentioned Search Head. It contains all the gathered event, and we can send search queries via it. It gives the possibility to visualize the data, and configure the Splunk units in the system, although we can do it by editing the proper configuration files.

3.6.1 Splunk vs Splunk free

The Splunk was chosen to be the SIEM of the network sensor, but like many other product it is not a freeware. Fortunately it has a free version, but it has restrictions and lacks some functions and features, so we needed some workaround. The missing functions and features [21]:

- 500 MB/day data uploading limit
- No alerting
- No TCP/HTTP forwarding
- No user roles
- No deployment server capabilities
- No Indexer clustering
- No distributed search
- No report acceleration

The 500 MB/day data upload is a limitation we can ignore, because this is only a test environment, so we will never exceed this limit.

The missing alerting function is probably the most hurting one. SIEMs' one of the most useful function is creating alerts. The operators can specify conditions for the events as alerts, so when these fulfil, the SIEM notifies the operators. There is no workaround, but at the demonstration we can search the specific events, so we can assume, if we had the paid version of the Splunk, we would have been notified about these events.

The free Splunk does not have the feature to forward any data to non-splunk software, so it excludes the possibility, to have a program, which can connect the Splunk and the Moloch. Fortunately there is a workaround, which is more complicated than sending a HTTP request. Splunk has an export function, when we can export search result in JSON format. The idea was that, when we export a result into a folder monitored by a script, that script can detect and read that JSON file, and opens a Moloch web interface in a browser with the corresponding data. This method is rather inconvenient, but still works with Splunk free.

Having only one user role is a very serious security risk, because everyone has administrator privileges, who logs in. But because this is only a test environment, we can ignore this.

Deployment server capabilities, index clustering, distributed search have meaning in a distributed system, but because we have only one indexer and one forwarder, these features' absence is not a problem.

Report acceleration makes the report creation faster, so another feature we can miss.

To summarize, the alerting and the TCP/HTTP forwarding are the missing useful features, but their absence only causes some inconvenience. Still the Splunk can index the events, and make statistics about them, so the Splunk free will be appropriate for this project.

3.6.2 Configuration of Splunk

For proper functioning we have to configure the Splunk forwarder and the indexer as well.

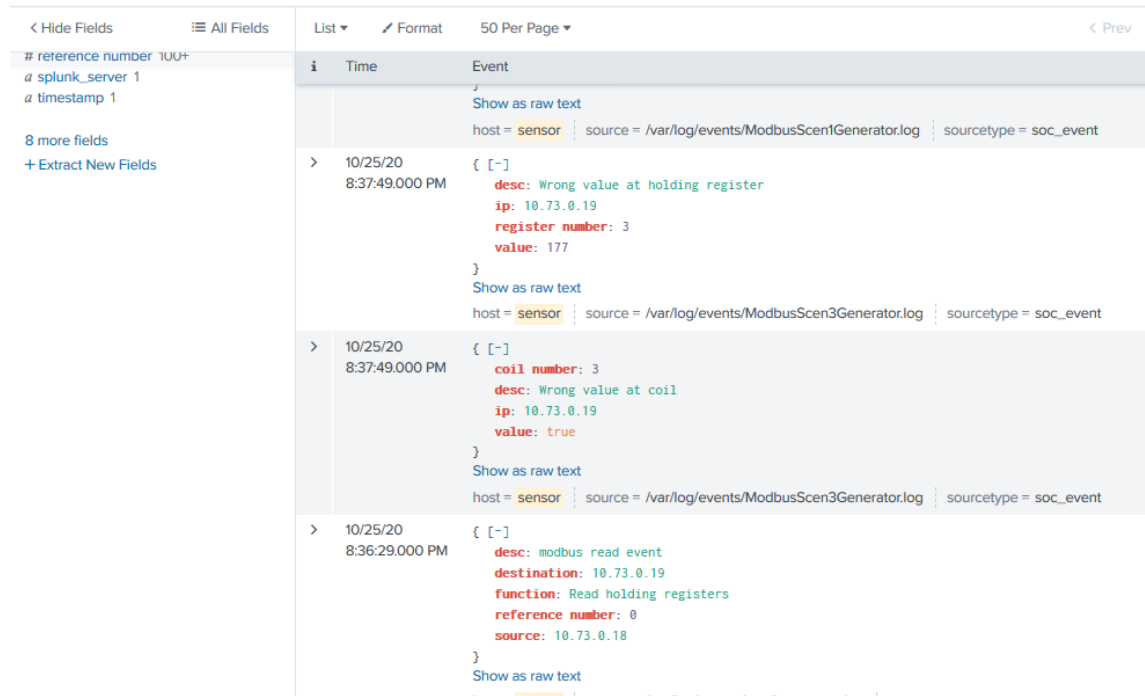
At the forwarder we have to set, which folder the forwarder must monitor, and what kind of event it should expect. The script will run on a machine with Linux based OS, so the `/var/log/events` folder is valid choice to write the log files, so it will be monitored. At the type of event I created a custom type `soc_event`, because I want to set the type's attributes as well. At the forwarder I set it to have JSON format, and I enabled the event breaker. This will be important, because that is how the indexer could distinguish events.

At the indexer I created a custom type with the same name `soc_event` and set some attributes as well. Here I disabled the line merge, so the indexer will not try to merge different line, I also set a line breaking symbol, which will be the event separator. This symbol is the newline character.

With this the Splunk forwarder will interpret the events correctly, and store them in a structured form.

3.6.3 Search and report

Search and report is an app in the Splunk, where the operators can request searches from the indexer, then here they also can process the result, like creating a chart from the result.



i	Time	Event
		Show as raw text host = sensor source = /var/log/events/ModbusScen1Generator.log sourcetype = soc_event
>	10/25/20 8:37:49.000 PM	{ [-] desc: Wrong value at holding register ip: 10.73.0.19 register number: 3 value: 177 } Show as raw text host = sensor source = /var/log/events/ModbusScen3Generator.log sourcetype = soc_event
>	10/25/20 8:37:49.000 PM	{ [-] coil number: 3 desc: Wrong value at coil ip: 10.73.0.19 value: true } Show as raw text host = sensor source = /var/log/events/ModbusScen3Generator.log sourcetype = soc_event
>	10/25/20 8:36:29.000 PM	{ [-] desc: modbus read event destination: 10.73.0.19 function: Read holding registers reference number: 0 source: 10.73.0.18 } Show as raw text

Figure 13: Example of a query

In the first picture we can see a query's result from Splunk. We can see the key value pairs. There are also additional information, like from which host the events came, or what is the name of the log file the events originated from, and what is the type of the event. Soc_event is a custom type, it help us to distinguish these events from other, even JSON, events.

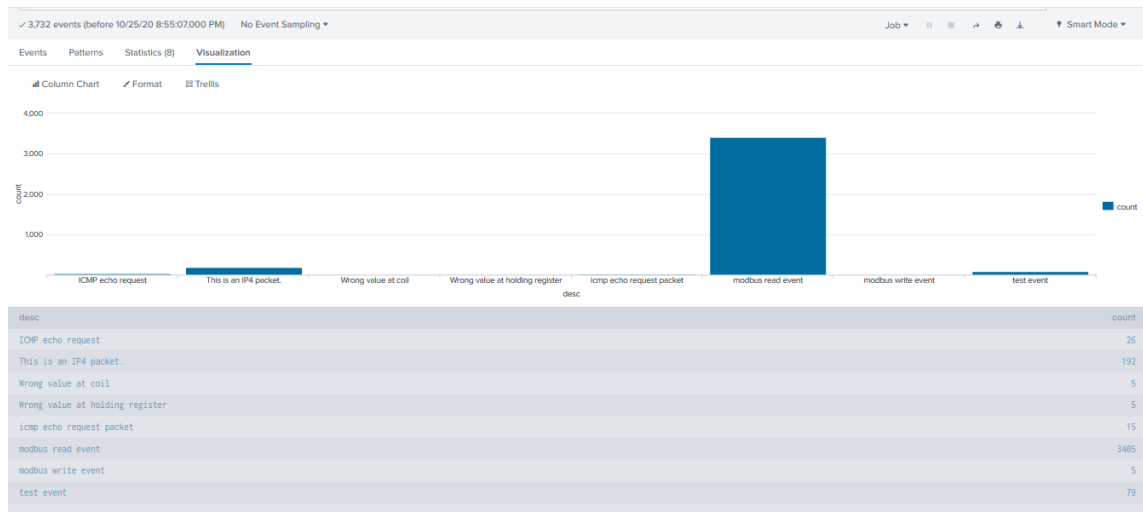


Figure 14: Example of a visualisation

In this picture we can see a result of a visualisation. We can query events and make different statistics and charts. In this example I queried all of the events from the sensor host and created a chart, where every column represents of the count of an event.

4 Evaluation of the system

The network sensor as a whole is complete, so utility scripts are missing, but the main function is ready to test. Although there were some testing before, but these were rather unit testing, here we will have attack scenarios, and the goal is to query their steps in the Splunk and detects their presence.

4.1 Test scenarios

There are two attack scenario, the first will be detected by the passive event generator, while the second will be detected by the passive and active generator as well. Both scenario starts with a searching, where they send ICMP ping requests to every address in the subnet. For good measure they will use arping as well, because there could be hosts, which ignores ICMP requests. Unfortunately Moloch ignores protocols, which are not over IP, so that is why the attacks has to use ICMP ping.

Next is the attack. The first script will steal information from the PLCs, while the second will manipulate its target's memory thus causing malfunction in the operation. Both target PLC is using Modbus protocols to communicate, so our attack scripts will use the "pymodbus" module to communicate with the PLCs.

4.2 Scenario 1

4.2.1 Attack script

In the first scenario the attack script first will scan for potential hosts. For this it sends ICMP echo requests to every addresses in the subnet. When it receives replies, it will know that, there is a host with that IP address. Next step is creating a Modbus client, and trying to create a Modbus connection with the host. Where the connection is successful, from there the script will read the memory of the target. It does not know the size of the memory, so it will read, until it receives an error message from the PLC. After it read everything it will go to the next host, and stop when there is no other.

For good measure this script will scan for hosts with arping as well. Arping is using ARP protocol instead of ICMP. It is better because ARP is a vital protocol to operate a network, while the ICMP is an utility protocol, so some hosts could ignore it. Normally arping would be used, but because Moloch ignores non-IP protocols, it will ignor ARP packets as well. So to demonstrate the host scanning step ICMP was used.

4.2.2 Generator scripts

We need two generator to detect all of the steps of the scenario 1 attack. Fortunately both was implemented, when I tested the passive event generator. We need a generator, which can detect ICMP echo requests and a generator, which can detect Modbus read requests.

ICMP event generator I did not have to modify anything in the original generator. It uses the expression function to query only the ICMP packets, then filter the ICMP echo requests. For every echo requests a event will be generated. The events contain the sender's and receiver's IP addresses as well.

Modbus event generator Fortunately the original Modbus generator is almost fitting for detecting scenario 1. I have just had to extend it to be able to detect and distinguish Modbus reading requests. An event contains what Modbus command was detected, who is the sender and the receiver, and which coil/register was written/read.

4.2.3 Evaluation

i	Time	Event
	8:18:09.000 PM	<pre> desc: ICMP echo request destination: 10.73.0.219 source: 10.73.0.18 } Show as raw text host = sensor source = /var/log/events/ICMP_event.log sourcetype = soc_event </pre>
>	10/25/20 8:16:06.000 PM	<pre> { [-] desc: ICMP echo request destination: 10.73.0.15 source: 10.73.0.18 } Show as raw text host = sensor source = /var/log/events/ICMP_event.log sourcetype = soc_event </pre>
>	10/25/20 8:16:06.000 PM	<pre> { [-] desc: ICMP echo request destination: 10.73.0.21 source: 10.73.0.18 } Show as raw text host = sensor source = /var/log/events/ICMP_event.log sourcetype = soc_event </pre>
>	10/25/20 8:16:06.000 PM	<pre> { [-] desc: ICMP echo request destination: 10.73.0.19 source: 10.73.0.18 </pre>

Figure 15: Querying events with description of "ICMP echo request"

As we can see in the picture, the sensor detected many ICMP echo requests from the same host.

Time	Event
10/18/20 11:35:41.000 AM	<pre>[-] desc: modbus read event destination: 10.73.0.216 function: Read holding registers reference number: 499 source: 10.73.0.18 } Show as raw text host = sensor source = /var/log/events/modbus_events.log sourcetype = soc_event</pre>
10/18/20 11:35:41.000 AM	<pre>[-] desc: modbus read event destination: 10.73.0.216 function: Read holding registers reference number: 498 source: 10.73.0.18 } Show as raw text host = sensor source = /var/log/events/modbus_events.log sourcetype = soc_event</pre>
10/18/20 11:35:41.000 AM	<pre>[-] desc: modbus read event destination: 10.73.0.216 function: Read holding registers reference number: 497 source: 10.73.0.18 } Show as raw text host = sensor source = /var/log/events/modbus_events.log sourcetype = soc_event</pre>
10/18/20 11:35:41.000 AM	<pre>[-] desc: modbus read event destination: 10.73.0.216</pre>

Figure 16: Querying events with description of "modbus read event"

In the picture we can see the Modbus read requests from 10.73.0.18. From the "reference" fields, we can see it read through the entire memory.

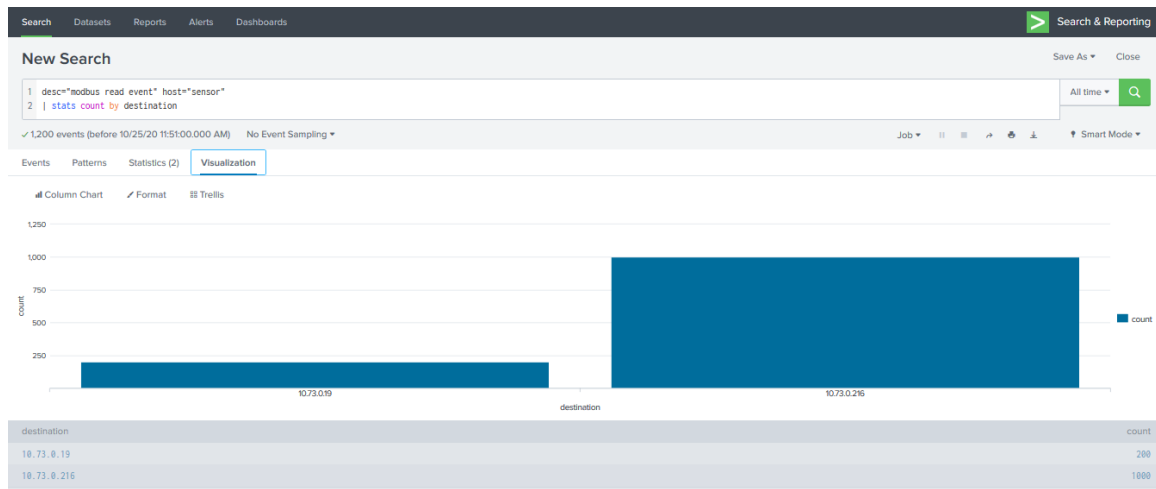


Figure 17: Statistic of the count of the read events at different hosts

In this picture we can see the result, when we query and group by the receiver's addresses. The PLC at 10.73.0.216 got more request, which means it has bigger memory space.

At the end the passive event generator was able to detect the steps of the first attack scenario, and the generated events are stored and available at the Splunk. So the first test was successful.

4.3 Scenario 2

4.3.1 Attack script

In the second scenario our goal is to test the active event generators too. The attack will start similarly like at the first one, it will scan for hosts. Here the attack will be targeted, so when it is convinced it can reach host with address 10.73.0.19, it will send write requests to the PLC. At the 10.73.0.19 host is a python modbus server mock, which was created for testing. The network has other real Modbus PLCs, and while reading is not a problem, writing could cause inconveniences to other students, who may working with these PLCs. The attack script will modify some coils and registers of the target Modbus server. It also inform us in the console, which coil and register was modified.

4.3.2 Generator scripts

In this scenario we have one passive event generator to detect ICMP and three active event generator, because I wanted to test the scheduler of the active event generator as well. I predefined that, the attack script could choose from 3 register and 3 coil to modify, thus every generator is checking on 1 coil and 1 register every minuets.

Modbus active event generator The only new generator for this test. It has 3 instance, with the difference of which coil and register they are querying. All three is using "pymodbus" module to create a Modbus client and send read request to the PLC. It knows what is the expected value of the inspected coil/register, and when they receive a different reply from the PLC, they create an event. This could be one or two event according to that, is the coil or the register or both has wrong value. The event contains, what has wrong value, what is the found value, the reference number of the coil/register, and the PLC's IP address, which is always 10.73.0.19 in this case.

4.3.3 Evaluation

The ICMP part of the attack was detected just like at the first scenario.



```
coil number 1 has been overwritten!  
register number 1 has been overwritten!
```

```
coil number 3 has been overwritten!  
register number 3 has been overwritten!
```

Figure 18: The modified coils' and registers' number

In the first picture we can see as the attack script inform us, which coil and register was modified.

i	Time	Event
>	10/25/20 8:38:09.000 PM	<pre> [-] desc: Wrong value at holding register ip: 10.73.0.19 register number: 1 value: 177 } Show as raw text host = sensor source = /var/log/events/ModbusScen1Generator.log sourcetype = soc_event </pre>
>	10/25/20 8:37:49.000 PM	<pre> [-] desc: Wrong value at holding register ip: 10.73.0.19 register number: 3 value: 177 } Show as raw text host = sensor source = /var/log/events/ModbusScen3Generator.log sourcetype = soc_event </pre>
>	10/25/20 8:38:09.000 PM	<pre> coil number: 1 desc: Wrong value at coil ip: 10.73.0.19 value: true } Show as raw text host = sensor source = /var/log/events/ModbusScen1Generator.log sourcetype = soc_event </pre>
>	10/25/20 8:37:49.000 PM	<pre> coil number: 3 desc: Wrong value at coil ip: 10.73.0.19 value: true } Show as raw text host = sensor source = /var/log/events/ModbusScen3Generator.log sourcetype = soc_event </pre>

Figure 19: The result of the attack in the Splunk

In the second picture we can see the corresponding active generators has detected the tampered registers and coils, so the active generator detected the attack, so the second test was also successful.

At the end both test concluded in success, so the main function of the network sensor was tested successfully.

5 Conclusion

5.1 Future work

The first possible continuation of the project is to write some more complex attacks, and detect them with the sensor system. I have planned a man in the middle attack, where the attacker with the help of the Bettercap tool gets between two PLC, who are communicating with each other, then eavesdrops their communication, and replays some messages. This is a more complex and definitely more interesting attack scenario. There could be more different attacks and generators as well.

Although the main function of the network sensor system is completed, there are some utility scripts, which have yet to be implemented. The first is the previously mentioned data archiving script, whose task is to send the PCAP file to an external database for later investigation.

The second is a connection script, whose goal is to connect the Moloch and the Splunk. When we query something from the Splunk, we do not see the packets, from which the events was generated. At this time the operator should have to open a Moloch web interface, and found manually the packets. This script should help in this. First I should extend the base generator script to automatically include the start and stop time in the events. Then when an operator wants the packets associated with the events, he just has to export these events to a folder monitored by this script, and it will query the packets from the Moloch and automatically open a web browser with the Moloch viewer.

The third script is the deployment scripts. It is multiple scripts, and their task is to make the installation of the system easy and automatic.

5.2 Result of the project

Our goal was to create a sensor system, which can detect steps of attacks, then provide information to the operators in ICS environment. The evaluation proved that, the 2 generator script can detect attacks. The generators are easily extendable and customizable too. Splunk and Moloch provided information about the events in the system, so the operators can examine them thoroughly. So in this aspect the project was successful.

Active event generation caused some challenge, because its generators could have great impact on the protected system. To mitigate that I had to find solution for that problem. I needed algorithms to schedule evenly the active event generators, and at the end I could create a working scheduler for this problem.

The other goal was to prove the concept of SOC with only using freeware software and our scripts. Moloch is a freeware software while Splunk has free version, which satisfies our needs. The scripts were written in Python and were using other freeware modules, so at the end we reached this goal as well.

This system can provide us with a customizable detection platform, but I would still not deploy in a real industrial facility with thousand PLCs and very complex network. For that kind of system Splunk free will not be enough, also detection is not the only security measure, which is needed to protect the facility. But if we see a small company with some hosts, servers and switches, there our system can work properly. There will not occur enough events to break the Splunk's 500 Mb/day constrain. Those companies also the ones, who do not have high budget, and probably cannot spend to much on security. For them the network sensor system is a great alternative cheap solution to monitor their system.

List of Figures

1	Structure of an average SCADA system	8
2	Structure of the system	16
3	Steps of the scheduler	25
4	Scheduling process	25
5	Steps of the dummy technique	29
6	Steps of the pumping technique	29
7	EDF without any technique	30
8	EDF with dummy	30
9	EDF with pumping	31
10	General approach result	33
11	Dense approach result	34
12	Rare approach result	35
13	Example of a query	38
14	Example of a visualisation	39
15	Querying events with description of "ICMP echo request"	41
16	Querying events with description of "modbus read event"	42
17	Statistic of the count of the read events at different hosts	42
18	The modified coils' and registers' number	43
19	The result of the attack in the Splunk	44

List of Tables

1	EDF without any technique: statistic	30
2	EDF with dummy: statistic	30
3	EDF with pumping: statistic	31
4	Difference between different amount of pumping	31
5	Generator counts of the different approaches	32
6	General approach statistic	33
7	Dense approach result statistic	34
8	Rare approach statistic	35

References

- [1] W. Y. Svrcek, D. P. Mahoney, B. R. Young, *et al.*, *A real-time approach to process control*. Wiley New York, 2006.
- [2] D. Krambeck, “An introduction to scada systems.” <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-scada-systems/>, 2015.
- [3] R. Shroff, “Difference between dcs and scada.” <https://control.com/thread/1365080091>, 2013.
- [4] W. Bolton, *Programmable logic controllers*. Newnes, 2015.
- [5] N. Falliere, L. O. Murchu, and E. Chien, “W32. stuxnet dossier,” *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, 2011.
- [6] R. Khan, P. Maynard, K. McLaughlin, D. Laverty, and S. Sezer, “Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid,” in *ICS-CSR*, 2016.
- [7] R. M. Lee, M. J. Assante, and T. Conway, “German steel mill cyber attack,” *Industrial Control Systems*, vol. 30, 2014.
- [8] D. Das, “An indian nuclear power plant suffered a cyberattack. here’s what you need to know.” <https://www.washingtonpost.com/politics/2019/11/04/an-indian-nuclear-power-plant-suffered-cyberattack-heres-what-you-need-know>, 2019.
- [9] K. I. CERT, “Threat landscape for industrial automation systems, h1 2019.” <https://ics-cert.kaspersky.com/reports/2019/09/30/threat-landscape-for-industrial-automation-systems-h1-2019>, 2019.
- [10] K. Stouffer, J. Falco, and K. Scarfone, “Guide to industrial control systems (ics) security,” *NIST special publication*, vol. 800, no. 82, 2014.
- [11] N. Lord, “What is a security operations center (soc)?.” <https://digitalguardian.com/blog/what-security-operations-center-soc>, 2019.
- [12] J. Petters, “What is a security operations center (soc)?.” <https://www.varonis.com/blog/security-operations-center-soc/>, 2020.
- [13] “Claroty platform.” <https://www.claroty.com/comprehensive-platform-overview>, 2020. For more information request documents from the vendor.
- [14] “Scadafence platform.” <https://www.scadafence.com/platform/>, 2020. For more information request documents from the vendor.
- [15] “Nozomi networks guardian.” <https://www.nozominetworks.com/downloads/US/Nozomi-Networks-Guardian-Data-Sheet.pdf>, 2020.
- [16] K. Jeffay, D. F. Stanat, and C. U. Martel, “On non-preemptive scheduling of periodic and sporadic tasks,” in *IEEE real-time systems symposium*, pp. 129–139, US: IEEE, 1991.

- [17] J. Y. Leung, *Handbook of scheduling: algorithms, models, and performance analysis*. CRC press, 2004.
- [18] “Pymodbus - a python modbus stack.” <https://pymodbus.readthedocs.io/en/latest/readme.html>, 2020.
- [19] “Client - functions.” <https://python-snap7.readthedocs.io/en/latest/client.html>, 2020.
- [20] “Splunk enterprise.” <https://www.splunk.com/pdfs/product-briefs/splunk-enterprise.pdf>, 2020. Brief solution of the Splunk Enterprise product.
- [21] “About splunk free.” <https://docs.splunk.com/Documentation/Splunk/8.1.0/Admin/MoreaboutSplunkFree>, 2020.