ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA STROJNÍ

DIPLOMOVÁ PRÁCE

2021

DAVID
JANATA

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Janata** Jméno: **David** Osobní číslo: **458483**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Průmysl 4.0**

Studijní obor: **bez oboru**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Analýza dat síťového provozu s ohledem na bezpečnost průmyslových aplikací s možnou podporou statistických metod a strojového učení

Název diplomové práce anglicky:

Analysis of network traffic data with regard to the security of industrial applications with the possible support of statistical methods and machine learning

Pokyny pro vypracování:

Navrhněte metodu a vytvořte prototyp nástroje (opensource) pro analýzu dat v průmyslových sítích za účelem detekce anomálií pro monitorování a vyhodnocování bezpečnostních rizik ve smyslu P4.0 (např. detekce možného kybernetického útoku na průmyslové zařízení, tj. linku, výr. stroj,...) .
Proveďte rešerši problematiky (zabezpečení průmyslových linek a strojů ve smyslu P4.0) a existujících metod, statistických analýz síťového provozu, metod strojového učení a neuronových sítí, a porovnejte je z hlediska úspěšnosti detekce s nově vytvořenými metodikami založenými na bázi umělé inteligence a případně i práce s big daty.
Součástí rešerše by měl být průzkum metod předzpracování dat a získávání důležitých atributů (statistické ukazatele, entropie, …).
Pokuste se zkoncipovat scenář možného útoku na zařízení a řešení co nejvčasnější detekce takového útoku. Pro data (uměle vytvořená i reálná) navrhněte metodu detekce nestandardních jevů v datech s podporou strojového učení (např. neuronových sítí).
Experimentálně vyhodnoťte vaše řešení a náležitě zdokumentujte.
Rozsah práce min. 50.01 stran + přílohy.
Grafický obsah max 55%.

Seznam doporučené literatury:

[1]M. Ghanbari and W. Kinsner, "Extracting Features from Both the Input and the Output of a Convolutional Neural Network to Detect Distributed Denial of Service Attacks," in 2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC), Berkeley, CA, Jul. 2018, pp. 138–144, doi: 10.1109/ICCI-CC.2018.8482019.
[2] DAVID TERRAZAS GONZALEZ, Jesus a Witold KINSNER. Zero-crossing analysis of Lévy walks for real-time feature extraction: Composite signal analysis for strengthening the IoT against DDoS attacks. In: 2016 IEEE 15th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC) [online]. IEEE, 2016, 2016, s. 143-153 [cit. 2021-04-11]. ISBN 978-1-5090-3846-6.
[3]S. McLaughlin et al., "The Cybersecurity Landscape in Industrial Control Systems," Proc. IEEE, vol. 104, no. 5, pp. 1039–1057, May 2016, doi: 10.1109/JPROC.2015.2512235.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Ivo Bukovský, Ph.D., U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.04.2021** Termín odevzdání diplomové práce: **13.08.2021**

Platnost zadání diplomové práce: _____

doc. Ing. Ivo Bukovský, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Miroslav Španiel, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání _____ Podpis studenta _____

Poděkování

Rád bych vyjádřil vděčnost všem, kteří mi radili a obětovávali svůj čas při vytváření této práce. Jmenovitě bych chtěl poděkovat doc. Ing. Ivu Bukovskému Ph.D. za jeho vedení. Dále Ing. Matěji Černému za jeho obětavost. A v neposlední řadě Ing. Petru Svobodovi Ph.D. za jeho čas.

Prohlašuji, že jsem práci vypracoval samostatně pod vedením doc. Ing. Iva Bukovského Ph.D., pouze s použitím literatury uvedené na konci práce.

…….

V Praze dne

…….

Podpis

Abstrakt: V práci je rozebírána bezpečnost industriálních IoT systému a jejich odolnost proti případným útokům. Dále jsou vysvětleny užitečné metody detekce anomálií a strojového učení. V praktické části jsou poznatky aplikovány na stažená data a poté i na data generovaná v reálném čase.

Klíčová slova: Strojové učení, detekce anomálií, kyber-bezpečnost, IoT

Abstract: This thesis concerns itself with cyber-security of industrial IoT devices and systems with regard to their ability to withstand possible attacks. Part is dedicated to explaining theory of anomaly detection methods and machine learning in general. This gained insight is then applied to a downloaded set of data and later to data generated in real-time.

Keywords: Machine learning, anomaly detection, cyber-security, Iot

ÚVOD

Téma diplomová práce je v průsečíku tří problematik. Jedná se o návrh a vývoj prototypu softwaru, takže by se dalo mluvit o disciplíně softwarového inženýrství, dále zkoumá problematiku počítačových sítí, přesněji řečeno industriálních počítačových sítí a v neposlední řadě se zabývá prací s daty.

Práce je rozdělena na dvě části. Část teoretická, ve které proběhne rešerše zmiňovaných problematik a jejich představení čtenáři. A dále část praktická, ve které získané poznatky nejprve aplikuji na staženou sadu dat za účelem detekce anomálií a poté budou algoritmy detekce anomálií použity jako část výsledného prototypu softwaru, který bude v reálném čase monitorovat komunikaci serveru s clientem. Na tuto komunikaci pak budou spouštěny simulace útoků, které by měl software spolehlivě odhalit.

1. PRŮMYSLOVÉ SÍTĚ A JEJICH ZABEZPEČENÍ

Právě probíhající čtvrtá průmyslová revoluce s sebou přináší značné výhody. Firmy, které úspěšně adoptují novinky z oblastí automatizace, internetu věcí či digitalizace, dosahují podle metastudie od McKinsey & Company [1] garantovaného zvýšení produktivity mezi 15 a 20 procenty. Tento přírůstek produktivity je primárně zapříčiněn snížením prostojů drahých výrobních zařízení a využíváním prediktivní údržby. Svoji roli však samozřejmě hraje i převod jednoduchých manuálních prací z člověka na stroj.

Zvýšená efektivita není však jediným přínosem robotizace a digitalizace továren. Velkým přínosem pro plánovací oddělení firem se stala také data z výroby. Informace o kvalitě výrobků, o rychlosti výroby, či o již zmiňovaných prostojích. Všechna data ze strojů spojených s internetem věcí, ukládaná v databázích, se dají analyzovat a výsledky analýz dále aplikovat a zlepšovat tím chod celého závodu. Díky detailnímu poznání životního cyklu výrobku, od objednávky až po expedici, může například plánovač výroby objednávat čím dál tím přesnější počty polotovarů, nástrojů, či jiných potřebných věcí. Dosahuje tak, díky obřím kvantům dat, stále štíhlejší výroby.

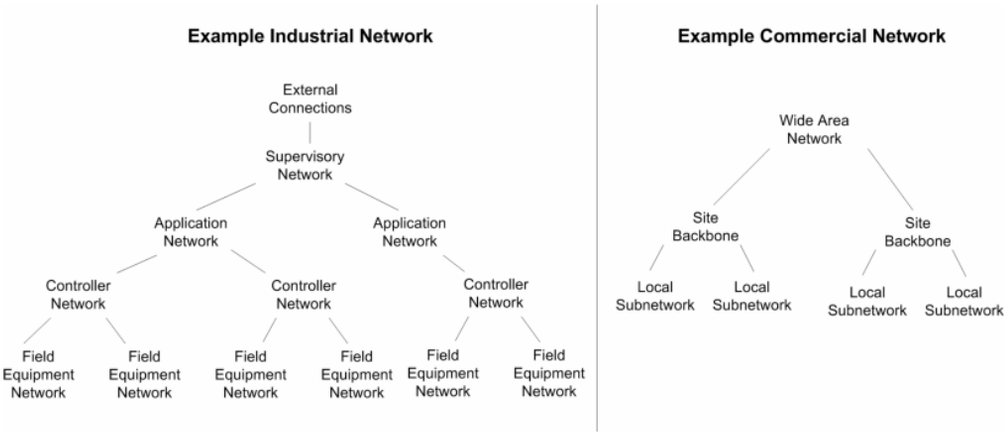
Nikdo nemůže popírat, že Průmysl 4.0 tedy zlepšuje kvalitu života zaměstnanců na všech úrovních firemní hierarchie. Od šéfa výroby, který může detailně monitorovat chody linek z pohodlí své kancelářské židle. Přes technologa, který má k dispozici stále přesnější analýzy. Až po obyčejnou obsluhu stroje, jejíž práce se stává fyzicky méně náročnou. Tyto jednoznačné benefity s sebou ale nesou i stinnou stránku věci. Pokud si odmyslíme stále nejasný sociologický dopad na společnost, je jako hlavním kontroverzním tématem s IoT spojována bezpečnost. Pro

upřesnění je myšlena kybernetická bezpečnost. Ve studii Cisco Annual Cybersecurity Report z roku 2018 [2] se bezpečnostní specialisté výrobních závodů nechali slyšet, že 31 % již vidělo pokus o útok na jejich IoT zařízení a 38 % ho očekává do roka. Ve spojení se zjištěním ze stejné studie, že 53 % úspěšných útoků resultuje ve ztráty vyšší než $500,000 pro postiženou společnost, se důležitost zabezpečení vnitřních i vnějších firemních síťových spojení stává zřejmou.

1.1. INDUSTRIÁLNÍ SÍTĚ A ŘÍDÍCÍ SYSTÉMY

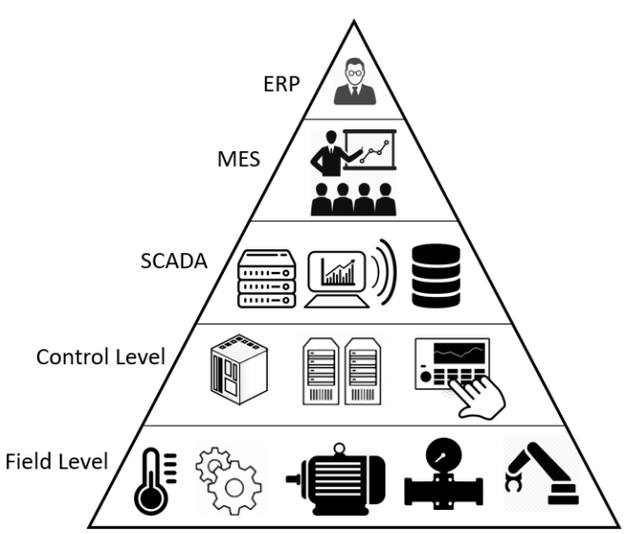
Komunikace v průmyslových sítích by se na první pohled mohla zdát velmi podobnou komunikaci v sítích komerčních (komerčními sítěmi myšleny například počítačové sítě v kancelářích). A samozřejmě mají mnoho společného, rozdíly jsou však v některých částech výrazné [3]. Hlavním rozdílem je fakt, že industriální sítě jsou připojeny, řídí a monitorují fyzické, reálné systémy. Tento rozdíl resultuje v různé požadavky na bezpečnost, rychlost a další atributy industriálních sítí oproti sítím komerčním.

Architektury těchto dvou systémů se také značně liší. Komerční sítě nejčastěji využívají více spojení typu LAN (Local Area Network), spojené jednou páteřní sítí, či WAN (Wide Area Network) po celém rozsahu. V kontrastu, i malé industriální sítě mají většinou několik vrstev. Sensory a motory ve vrstvě nejnižší, dále jejich ovladače nad nimi. Vrstvu nad ovladači může zaplňovat například HMI, které do vrstvy nad sebou posílá data. Obrázek 1 - Architektury sítí rozdílné architektury ilustruje graficky.



Obrázek 1 - Architektury sítí [3]

Kyber-bezpečnost industriálních sítí bývala v minulosti jednodušší problematikou než dnes. Snahou velkých společností o standardizaci a konektivitu industriálních sítí se sítěmi komerčními, se začaly adoptovat například praktiky jako používáních klasických počítačů ve výrobě a jejich propojení s externími komerčními sítěmi. Dále začali výrobci industriálních ovladačů postupně přecházet na známější a více rozšířené operační systémy. Toto samozřejmě zvýšilo synergii mezi výrobní halou a kanceláří, snižuje to však faktor izolace od okolního světa pro industriální sítě [4]. V praxi se tím dále zvyšuje potřeba zabezpečení takto propojených sítí.



Obrázek 2 - SCADA Pyramida

Dále se pobavme o komunikačním protokolu OPC UA. Relativní novinka na trhu, která si postupně získává přízeň stále větší části industriálních podniků. OPC UA, neboli OPC Unified Architecture, je silně univerzální komunikační protokol, pokus o standardizaci dříve silně diverzního trhu. V 80. letech 20. století se prudce začala zvyšovat poptávka po industriální softwarové automatizaci. Zvýšení poptávky zákonitě vedlo ke vzniku různých společností, jež by dodávaly nutný hardware. Nedostatečná domluva či nevůle vedla k tomu, že jednotlivé hardwarové komponenty od různých společností, všechny využívaly jiné komunikační protokoly, a proto byla práce s nimi velmi náročná a špatně do budoucna škálovatelná. Proto se v roce 1996 tyto různé společnosti rozhodly společně založit nadaci OPC, ve snaze zjednodušit spotřebitelům používání jejich produktů. Nejprve byl vydán první standard s názvem OPC, který se postupně zdokonaloval, až v roce 2008 byl vydán pod novým názvem „OPC UA". K dnešnímu dni má nadace OPC přes 750 členů od malých firem až po světové giganty na trhu s automatizací. OPC UA se stále zdokonaluje, a své směřování cílí primárně na bezpečnost a škálovatelnost systémů jeho uživatelů.

V neposlední řadě je třeba zmínit, že komunikace přes protokol OPC UA, stejně jako přes většinu protokolů, spočívá ve výměně takzvaných „packetů". Packet má dvě části, a to z část řídící (metadata) a část uživatelskou (anglicky zváno „payload"). Řídící část packetu obsahuje nutné informace ke správnému doručení packetu. Lze přirovnat k obálce dopisu s adresou. Uživatelská část obsahuje zašifrované posílané informace, tj. samotný dopis.

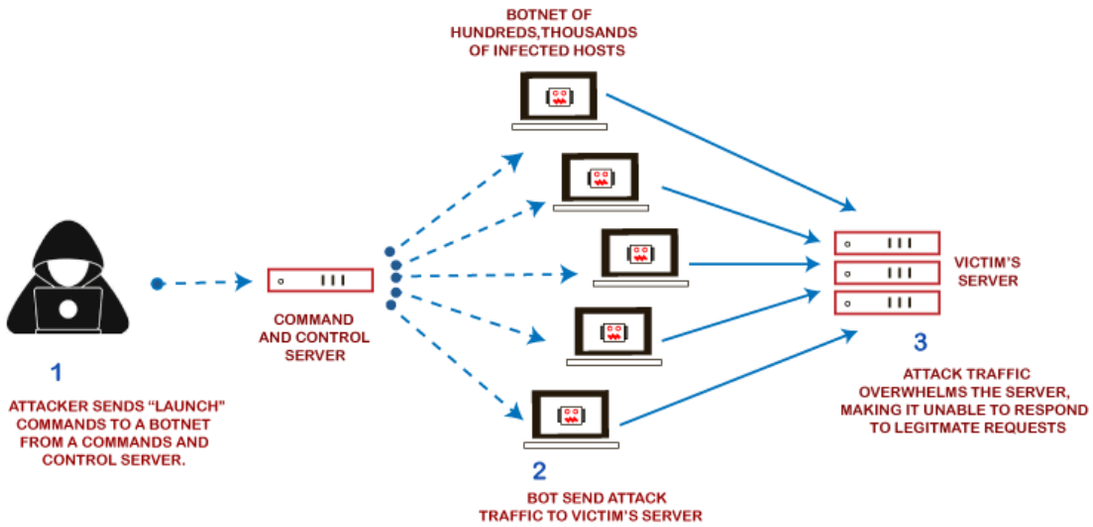
1.2. TYPY ÚTOKŮ A JEJICH PREVENCE

V následujícím textu se čtenář seznámí s různými typy útoků, kterým musí IoT čelit. Bude vysvětlen jejich princip, uvedeny příklady z reálného světa a současné trendy v obraně proti nim.

1.2.1. BOTNET

Botnetem se v informatice nazývá skupina počítačů spojená přes internet, na nichž běží instance stejného „bota" (internetového robota). V našem případě se botnetem myslí skupina počítačů infikovaných virem, o kterém většinou majitel počítače neví. Prakticky poskytuje tvůrci viru možnost využívat infikované počítače k libovolným účelům. Tyto robotí farmy mají pro hackery několik využití. K méně závažným se řadí například posílání e-mailového spamu ze schránky infikovaného počítače či těžení kryptoměn, které v nejhorším případě postupně znehodnotí grafickou kartu majitele hardwaru. K více závažným by se dal zařadit spyware, který posílá tvůrci viru informace z infikovaného stroje. Informace jako hesla, čísla kreditních karet či osobní údaje, které se dále prodávají na černém trhu. Nejčastější využití však najdou takto ovládané stroje jako „pěšáci" v útocích typu DDos.

DDoS neboli „Distributed Denial-of-Service" je typ útoku, kterému experti přes kyber-bezpečnost věnují velkou část své pozornosti. Jednoduše vysvětleno, při tomto typu útoku se všechny infikované počítače pokouší zahltit cíl útoku, většinou jeden nebo více webových služeb, s veřejnou IP adresou. Při úspěšném útoku je šířka síťového pásma cíle kompletně zaplněna požadavky od botnetu a cíl tak není schopen poskytovat dál službu opravdovým uživatelům. Na Obrázek 3 – Diagram DDoS útoku si může čtenář prohlédnout srozumitelný diagram útoku typu DDoS.



Obrázek 3 – Diagram DDoS útoku [5]

V žebříčku nejznámějších botnetů se jen málokdo blíží malwaru s názvem Mirai. Poprvé objeveno v roce 2016, toto dílo skupiny vysokoškoláků, primárně útočilo na špatně zabezpečené IoT zařízení. Zařízení typu digitální hodiny, termostaty či bezpečnostní kamery a domácí routery byly infikovány a když jejich počty stouply do vyšších statisíců byly použity v útocích. Dvacátého prvního října 2016, Mirai úspěšně vypnul služby poskytovatele DNS jménem Dyn. Toto vedlo k zamezení přístupu k obsahu stránek jako například Twitter, Netflix, AirBnB a další [6]. Jindy byl Mirai za výpadkem internetu více než 900.000 občanů Německa [7].

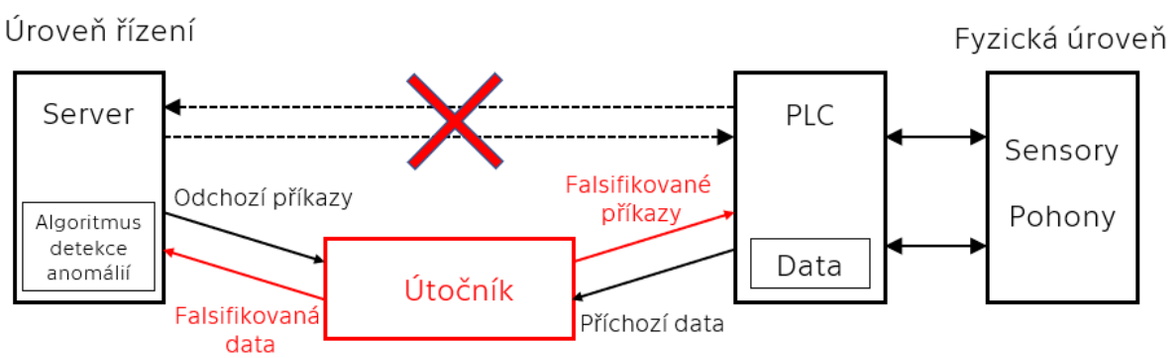
Pozornému čtenáři je tedy jasné, že DDoS útoky v principu nemohou způsobit materiální škody. Hardware nemůže být nijak poničen a software ani data nemohou být vymazána. Takovýto útok na IoT zařízení ve výrobních továrnách může však vést k dočasnému kolapsu výroby. Takto způsobené prostoje mohou resultovat ve zhoršení reputace společnosti či dokonce k pokutám za pozdní dodání objednávky.

Jak se proti DDoS bránit?

Obrana proti DDoS útokům má dva zřejmé cíle. Prvním z nich je zajištění přístupu pro opravdové uživatele služby. V našem případě zajištění plynulého toku informací mezi servery a IoT zařízeními. Druhým cílem je prevence zaplnění šířky pásma našeho systému. Na pořadí cílů záleží, hlavním cílem je udržet dostupnost pro legitimní uživatele. Jak tedy tyto dva cíle naplnit? Zde vstupuje detekce anomálií. Kvalitně nakonfigurovaný software dokáže rozpoznat anomálie v síťovém provozu pomocí vzorců obvyklého chování v kombinaci s reputací IP adres, které žádají o přístup. Více ke strategii a technikám obrany v praktické části.

1.2.2. MAN-IN-THE-MIDDLE

Méně častým, potenciálně však nebezpečnějším typem útoku je takzvaný Man-in-the-Middle. Přeloženo z angličtiny „Muž uprostřed". Při MitM útoku se hacker snaží nabourat Server-Client komunikaci a posílat upravené či fabrikované informace.



Obrázek 4 - Diagram útoku Man-in-the-Middle

Na Obrázek 4 - Diagram útoku Man-in-the-Middle je přehledně ilustrován princip takového útoku. Útočník, vydávající se z pohledu severu za clienta, obdrží jako první veškeré příkazy, které pak může libovolně upravovat, či nepouštět dál. Z druhé strany se clientovi jeví útočník jako server, kterému posílá veškerá data.

Jak se proti Man-in-the-Middle bránit?

Při obraně pro MitM se opět využívají principy detekce anomálií. Jednoduché přístupy se v tomto případě většinou snaží pouze identifikovat, zda k útok nastal. Toho se většinou dosahuje využitím strojového učení. Model monitoruje běžné dění v server-client komunikaci a pokud se útočník pokouší do proudících dat či příkazů zasahovat, model upozorní obsluhu na možný útok. Tento přístup je však z pohledu bezpečnosti informací nedostatečný, jelikož dokáže odhalit pouze útočníka, který se snaží ovlivňovat komunikaci. Pouhý odposlech tento přístup není schopen odhalit. Často se proto doporučuje tento model rozšířit o časovou složku [8]. Normálnímu packetu informací totiž vždy trvá výměna zhruba stejnou dobu. Když se do spojení napojí útočník, musí každý packet sám přijmout, rozšifrovat, případně upravit, opět zašifrovat a poslat dál. Tento akt značně prodlužuje dobu cesty packetu. Dobře nastavený algoritmus detekce anomálií je tyto diskrepance schopný včas odhalit a informovat obsluhu. Podrobnosti a další možnosti obrany proti MitM budou probrány v praktické části práce.

1.2.3. N-DAY EXPLOIT

Pro vysvětlení principu N-Day exploit si nejdřív vysvětleme častěji slýchanou verzi takového útoku, tak zvaný Zero-Day exploit. V informatice je „zneužití nultého dne" označením pro hrozbu, která využívá zranitelností v softwaru, která není veřejně známá. Označení „nultý den" vychází právě z nevědomosti vývojářů takto ohroženého softwaru. Nula dny je myšlen počet dní, který měli developeři na opravu bezpečnostní hrozby ve svém programu. Nula je to právě proto, že jim hrozba není známa. [9] Vývoj takového viru, který využívá nedokonalosti zabezpečení softwaru je nákladný. Hacker musí důkladně projít celý zdrojový kód napadaného programu, aby byl schopný chybu najít a zneužít. Cíli takto nákladných

operací bývají zpravidla vládní organizace, velké korporáty a ve výjimečných případech byly zjištěny instance, kde vládní organizace tímto způsobem kyberneticky útočily na své mezinárodní nepřátele.

Proslulým příkladem takového útoku je Stuxnet [10], kdy se měly složky Israelského zpravodajského sboru pomocí počítačového červa dostat do SCADA systému Iránského závodu na obohacování uranu v Natanzu. Poté co hackeři převzali kontrolu nad PLC, měnili výstupní frekvence frekvenčních měničů tak, aby znehodnotili drahé industriální odstředivky.

„Zneužití N-tého dne" je v porovnání se známějším „zneužitím nultého dne" často opomínanou hrozbou. V industriálních komplexech je však mnohdy hrozbou reálnější. V porovnání s prostředím klasického IT sektoru, trvá vydání opravné aktualizace vývojářům softwarů řídících systémů několikanásobně delší dobu. A to i v situacích kdy je zranitelné místo v softwaru známo desítky až stovky dní. Důvody pro takto pomalé jednání jsou četné, podle zakladatele a ředitele společnosti Red Balloon Security jsou čtyřmi hlavními právě tyto [11]:

1. Software řídícího systému musí vždy být online. Přestávky jsou pro velké společnosti extrémně nákladné.
2. Neexistence jednotných standardů. Na rozdíl od běžných uživatelských softwarů, opravné aktualizace industriálních řídících softwarů jsou většinou manuální prací. Výrobní továrny navíc mnohdy používají více softwarů od více výrobců. Například u výrobců PLC to vede k nutnosti znát detailně postupy aktualizace u každého z nich.
3. Opravné aktualizace se mezi dodavateli, kteří používají sdílený kód, šíří jen zřídka. Existují instance, kdy byla zranitelnost nahlášena dodavateli v telekomunikačním sektoru, byla opravena dodavatelem softwaru Intel, ale aktualizace nebyly aplikovány řadou dalších velkých dodavatelů v oblasti řídících systémů, tudíž se pro výrobní podniky riziko, ani po aktualizaci jednoho ze softwarů, nesnížilo.

4. Prodloužení životnosti softwaru. Řídící systémy jsou v praxi používány dlouho po ukončení podpory ze strany vývojáře. Vývojáři, kteří chtějí prodávat nové produkty, nejsou motivováni vydávat opravné aktualizace, přestože jsou jejich softwary v praxi stále široce používány.

Jak se proti N-Day exploits bránit?

Forma obrany proti takto spletité hrozbě není jednoznačná. Zřejmým problémem je nutnost spolupráce více stran při hledání řešení. Ze strany uživatele technická neproveditelnost vývoje vlastní aktualizace, a ze strany vývojáře neochota či pomalý vývoj aktualizace. I přes tyto překážky lze identifikovat hodně nedostatků v běžném užívání chybných softwarů a tyto nedostatky postupně odstraňovat. Hlavním doporučením je, aby se pro uživatele industriálních řídících systému stala bezpečnost prioritou. Při vyjednávání koupě s vývojáři by měli uživatelé plně trvat na robustních vestavěných bezpečnostních prvcích. Každá jednotlivá komponenta v řízeném systému by měla sama v sobě obsahovat co možná nejlepší zabezpečení. V praxi toto není běžné, a pokud se má bezpečnost průmyslových sítí do budoucna zvyšovat, je adopce takových to priorit na obou stranách nutná. Druhým doporučením by byl proaktivní přístup uživatelů k vlastní kyber-bezpečnosti. Pravidelná školení bezpečnostních techniků o nově objevených zranitelnostech v jimi používaných softwarech by měla být součástí každé společnosti. Proškolení technici by měli dále ochotně aplikovat vydané opravné aktualizace. Zranitelná místa, pro která vývojář zatím nevydal (či vydávat nebude) opravné aktualizace, by měli technici co nejlépe izolovat od zbytku systému a rigorózně monitorovat. Takto ohrožené části systému by měly podléhat silným bezpečnostním pravidlům, jako například zákaz používání datových nosičů.

2. STROJOVÉ UČENÍ

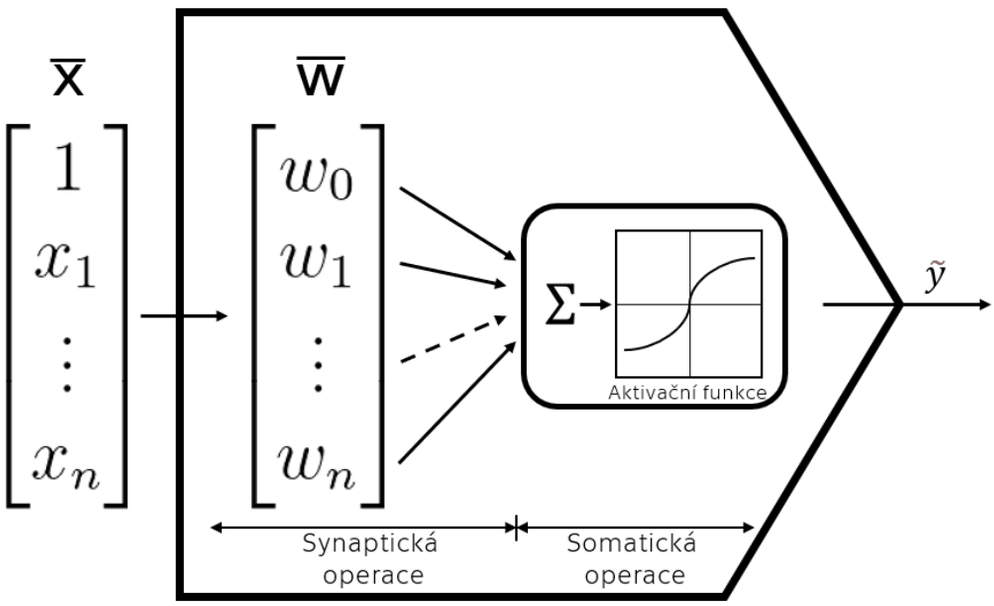
V následujícím textu bude čtenáři představena problematika strojového učení. Čtenář bude detailně uveden do základních konceptů matematiky skrývajících se za strojovým učením. Vysvětleny budou i různé druhy neuronů a neuronových sítí, jejich výhody a aplikace. Dále budou představeny statistické nástroje pro snadnější zpracování dat a jejich návaznost na přesnost výsledku. V neposlední řadě si ukážeme různé učící algoritmy a vybrané metody detekce anomálií.

2.1. NEURONOVÉ SÍTĚ

Jak již název napovídá, inspirací pro umělé neuronové sítě jsou neurofyziologické neurony v nervových soustavách živých bytostí. Za počátek vzniku oboru neuronových sítí je považována práce Warrena McCullocha a Waltera Pittse z roku 1943 [12].

2.1.1. ARCHITEKTURA NEURONOVÝCH SÍTÍ

Základní stavební jednotkou neuronových sítí je nepřekvapivě neuron. Výstupy jednotlivých neuronů mohou dále figurovat jako vstupy do neuronů dalších, vzniká tzv. neuronová síť. Umělý neuron má, v paralele s neuronem fyziologickým, schopnost se učit z předchozích zkušeností (hodnot) a adaptovat se tak, aby z něj vystupovaly stále přesnější informace (odhady).



Obrázek 5 - Architektura neuronu, překresleno z [13]

Obrázek 5 - Architektura neuronu, překresleno z zjednodušeně ilustruje vnitřní fungování umělého neuronu. Vstupem je vektor $\bar{x}$ který je synaptické části vynásoben vektorem $\bar{w}$. Na obrázku 4 je toto ilustrováno řeckým písmenem velké sigma, a to právě proto že celou operaci matematicky zapíšeme:

$$f = \sum_{i=0}^{n} x_i \cdot w_i \qquad (1)$$

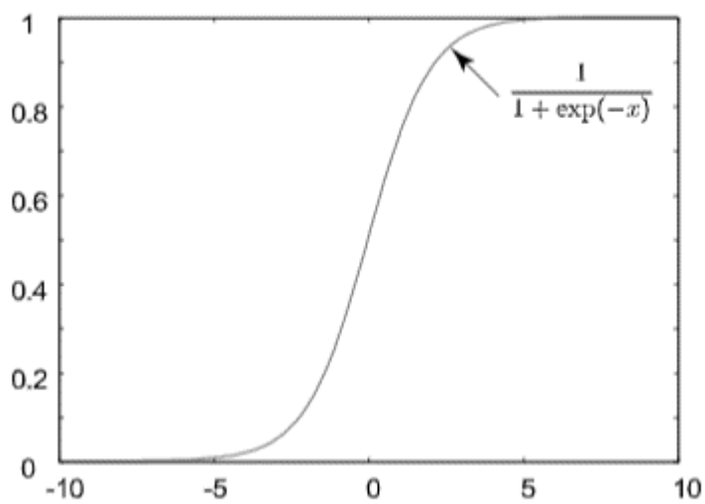
Pozn.: Jednoduché vektorové násobení, ukázané v rovnici (2) není jediným způsobem kterým lze synaptickou operaci provádět. Takovémuto způsobu se říká LNU – „Linear Neural Unit", neboli lineární neuron. Dalšími používanými způsoby jsou QNU a CNU, „Quadratic Neural Unit" a „Cubic Neural Unit" [14].

Na obrázku 4 jsme si vysvětlili levou stranu neuronu, tzv. synaptickou operaci. Dále si představíme pravou stranu, somatickou operaci. Ta upravuje výstup z neuronu na použitelný či užitečnější tvar. Aktivační

funkce z obrázku 4 připomíná sigmoidu, toto není náhodou. Její matematický předpis vypadá následovně:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Sigmoida je nejčastěji používanou aktivační funkcí. Vyniká hned několika vlastnostmi. Velkým přínosem je její nelinearita, to že jejím oborem hodnot je interval $\mathcal{H}_f = (0,1)$ či její strmost na intervalu $< 2,2 >$, která ji dělá užitečnou při binárních pravděpodobnostních klasifikacích.



Obrázek 6 - Sigmoida, převzato z [13]

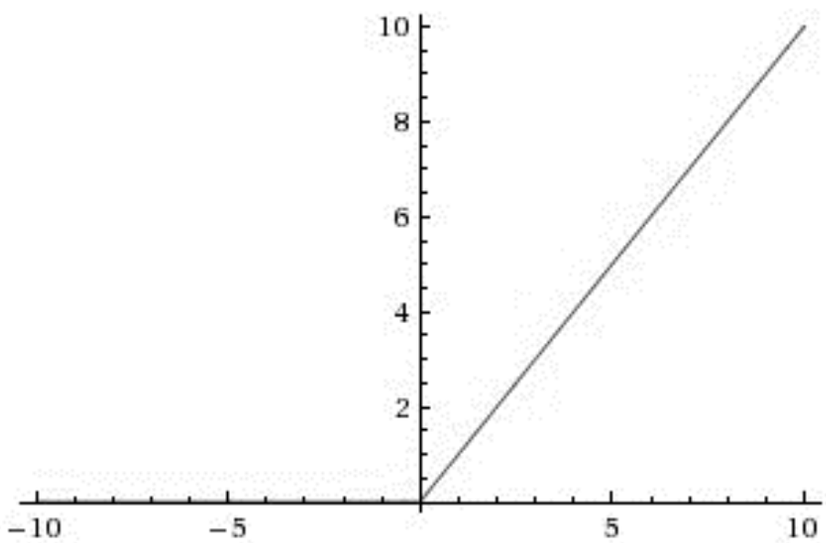
Mezi další používané aktivační funkce bychom mohli zařadit funkci lineární, ta však své uplatnění nachází hlavně ve formě aktivační funkce „ReLU" – Rectified Linear Unit. Důvodem pro používání bilineárního ReLU, místo klasické lineární aktivační funkce, je paradoxně jeho nelinearita. Klasická lineární funkce typu $y = \alpha \cdot x + \beta$, není pro opakovanou aplikaci v neuronových sítích ideální. Představme si situaci, kdy máme síť se vstupem, dvěma skrytými vrstvami a výstupem. Pokud bychom pro každý neuron použili striktně lineární aktivační funkci, byl by výsledek akorát lineární kombinací vstupů [15], tudíž bychom nepotřebovali více vrstev

neuronů, stačila by jedna. ReLU si ponechává linearitu v kladných hodnotách x, dává však nulu v hodnotách záporných.

Matematicky zapsáno:

$$f(x)_{ReLU} = max(0, x) \quad (3)$$

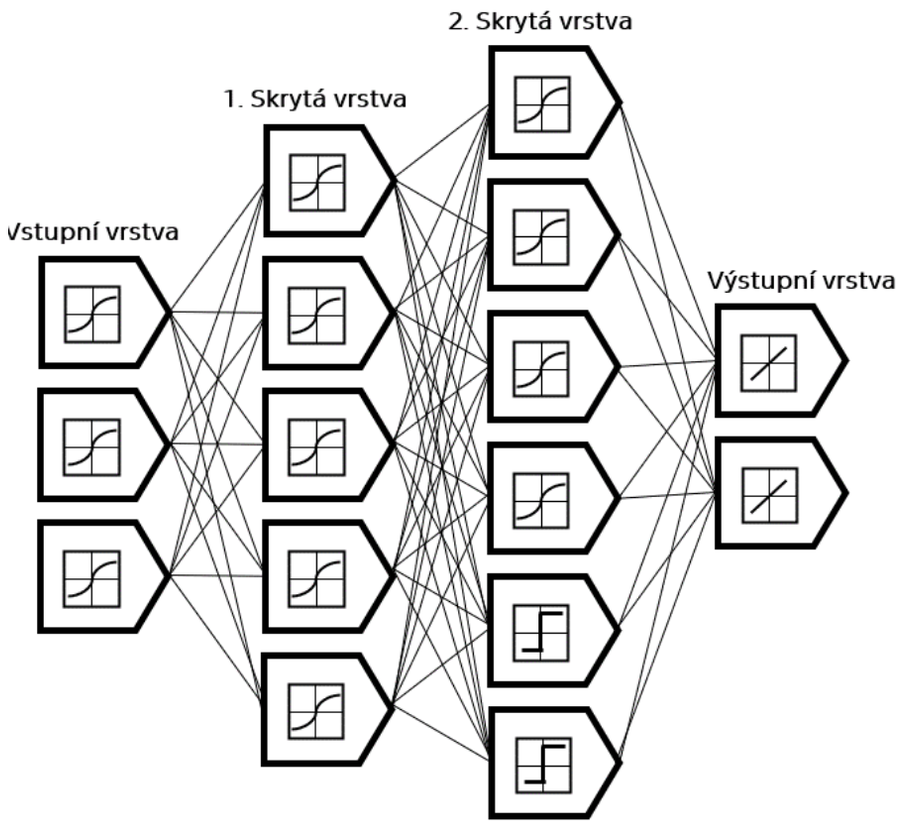
Graficky znázorněno:



Obrázek 7 - Aktivační funkce "ReLU" – převzato z [15]

Tato drobná úprava oproti klasické lineární aktivační funkci má na užitečnost dalekosáhlý dopad.

Neuronové sítě pak staví jednotlivé neurony za sebe tak, že kombinace výstupů z neuronů v jedné vrstvě, tvoří vstup do neuronů ve vrstvě další. První vrstva v pořadí se v praxi nazývá „input layer" – vstupní vrstva. Poslední vrstva neuronů je „output layer" – výstupní vrstva. Všem vrstvám mezi nimi se říká „hidden layers" – skryté vrstvy.



Obrázek 8 - Ilustrace neuronové sítě

2.2. PŘEDZPRACOVÁNÍ DAT

Jak bylo zmíněno v kapitole 4.3. do samotného neuronu vstupuje vektor x. Vstupní vektor (feature vector) reprezentuje různé atributy zkoumaných dat v matematické, počítačem snadno pochopitelné, formě. Příkladem takového vstupního vektoru může být RGB popis barvy v problematice zpracovávání obrazu. Takový vektor $x = [\mathrm{R(red)}, \mathrm{G(green)}, \mathrm{B(blue)}]$ má tedy pak tři složky, a to numerické vyjádření podílu červené, zelené a modré na finální podobě barvy.

Samotné předzpracování dat je nutným a neopomenutelným krokem v práci se strojovým učením. Předzpracování (preprocessing) se primárně provádí, protože měřená data z reálného světa nejsou v podstatě nikdy dokonalá. V reálných měřených datech se zpravidla vyskytuje nějaký šum,

místy chybí hodnoty a také jsou často ve formátu, který není přímo použitelný pro modely strojového učení. Toto „čištění" dat tedy bývá nevyhnutelné a jako přidanou hodnotu to zvyšuje přesnost a účinnost aplikovaných modelů.

Proberme tedy postupně některé možné situace a jejich řešení.

2.2.1. CHYBĚJÍCÍ DATA

Snad nejčastějším problémem, se kterým se v práci s reálnými daty setkáme, jsou chybějící záznamy. Příkladem tohoto může být chvilkové selhání sensoru na stroji, který měříme. Jindy se jedná o lidskou chybu při vyplňování excelové tabulky.

Převládajícím názorem na téma „jak řešit chybějící data" je, že má analytik dvě možnosti. Vymazat celou řadu nebo sloupec, ve kterém se chyba vyskytuje je první z nich. Toto řešení spolu nese samozřejmě nevýhodu v tom, že výsledkem takové úpravy je pak menší dataset a může to vést ke ztrátě důležitých dat. Druhou nabízenou možností je pak nahrazení chybějícího údaje střední hodnotou z údajů ostatních. Tato možnost je samozřejmě časově náročnější, ale zpravidla dosahuje vyšší přesnosti modelu.

2.2.2. ŠKÁLOVÁNÍ HODNOT

Škálování hodnot je můj volný překlad anglického názvu „feature scaling". Tento proces lze zařadit mezi základní operace předzpracování dat. Motivací k jeho aplikaci je umožnění (či zjednodušení) práce s daty, které obsahují široké spektrum hodnot či fyzikálních jednotek. Pokud by náš model obsahoval jeden vstup v gramech, další v tunách a třetí v litrech, mohlo by se stát, že čistě kvůli velikému rozdílu v nominálních hodnotách by docházelo k neopodstatněnému zkroucení vah. Je nutné podotknout, že některé algoritmy strojového učení jsou k veliké variaci hodnot náchylnější

než jiné. Například u algoritmů zakládajících se na Gradient Descentu je téměř nevyhnutelné předzpracování dat. Stejně tak tomu je u třídy algoritmů zakládajících si na vzdálenosti (myšleno například Euklidovská vzdálenost), jako je třeba K-Means. Naopak algoritmy typu „rozhodovací strom“ (decision tree) jsou vůči negativním dopadům různých rozsahů nominálních hodnot dat v podstatě imunní.

Feature scaling má mnoho podob a výběr správné metody pro stavěný model je důležitou, i když často opomínanou, součástí práce. Vysvětleme si tedy dvě nejrozšířenější metody škálování hodnot.

Normalizace dat

Známé také jako MinMax Scaling je operace s daty taková, že výsledkem je matice (vektor) se stejnými rozměry jako matice (vektor) vstupní. Normalizace dat je určena vzorcem:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad (4)$$

Normalizovaná data se pak pohybují pouze v intervalu <0, 1> tak, že původní nejnižší hodnota má po transformaci právě hodnotu 0 a původní nejvyšší hodnota má po transformaci právě hodnotu 1.

Standardizace dat

Známé také jako z-score je operace s daty taková, že výsledkem je matice (vektor) se stejnými rozměry jako matice (vektor) vstupní. Standardizace dat je určena vzorcem:

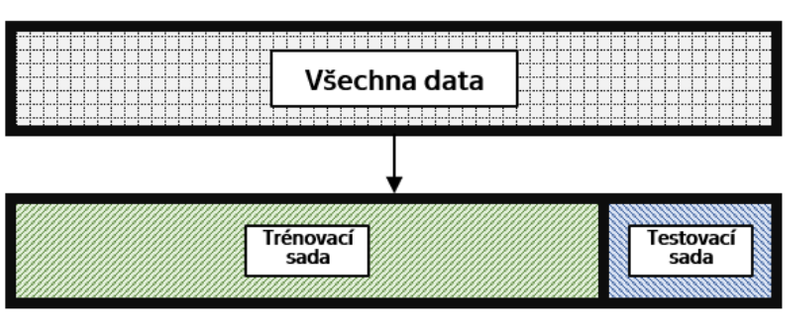
$$X' = \frac{X - \mu}{\sigma} \qquad (5)$$

Kde $\boldsymbol{\mu}$ je střední hodnota dat a $\boldsymbol{\sigma}$ je směrodatná odchylka. Standardizovaná data nejsou soustředěna do žádného pevně daného intervalu, mají ale vlastnosti takové, že hodnota, která byla původně právě hodnotou střední, je v transformovaných datech právě 0. Transformovaná data si drží stejné rozdělení jako data původní a mají jednotkovou směrodatnou odchylku.

Vyvstává tedy otázka, kdy aplikovat kterou z těchto dvou nabízených metod. Základním pravidlem pravé ruky může být původní rozdělení dat. V praxi se na data, která dodržují normální (Gaussovo) rozdělení, většinou používá metoda standardizace. Naopak u dat, která tomuto rozdělení nepodléhají, se doporučuje používat normalizaci. Jsou samozřejmě i další faktory, které toto rozhodování ovlivňují. V našem případě to může být například volba typu neuronu, který figuruje v naší síti. V případě kvadratického neuronu pak nedává moc smysl požívat data standardizovaná, jelikož se skrze kvadratický neuron stanou záporné hodnoty kladnými.

2.2.3. ROZDĚLENÍ DAT NA SUB-SETY

Abychom správně natrénovali náš model, je potřeba mít dostatečně velkou sadu dat, ze kterých se bude učit. Jak je graficky znázorněno na Obrázek 9, ne všechna data, která máme k dispozici, mají být použita pro naučení našeho modelu.



Obrázek 9 – Grafické znázornění rozdělení sady dat

Je logicky nutné náš model po naučení otestovat. Proto se před samotným začátkem učení rozdělí sada všech dostupných dat na dvě menší. A to na sadu trénovací a na sadu testovací. Obecně platným pravidlem je, že bychom měli data dělit zhruba v poměru 70:30 až 80:20 trénovací ku testovacím. Tento poměr je však možné měnit na základě vlastností, jako například tvar či velikost, dat používaných.

2.3. GRADIENT DECSENT

Gradient descent (dále jen GD) je základním algoritmem v disciplíně optimalizace. Jako forma zpětné propagace chyby, ve statistice, strojovém učení a datovém inženýrství, je první a nejčastější volbou při zápasech s optimalizačními úlohami. Jednoduchými ukázkami jeho použití jsou například: lineární regrese, logistitcká regrese, t-SNE clustering a mnoho dalších.

Premisou GD je hledání minima funkce, popisující optimalizovaný systém. K této funkci budeme dále referovat jako k cílové funkci $\mathbf{C_F}$ (cost function). K tomuto minimu lze algoritmem GD dojít inkrementálně, kroky ve směru záporné hodnoty gradientu cílové funkce.

Intuitivní vysvětlení GD podal Sagar Mainkar [16] ve svém článku pro web www.towardsdatascience.com. Mainkar v něm algrotimus přirovnává k cestě z hory do údolí s páskou přes oči. Jedinou pomůckou, kterou můžete využít, je přístroj, který Vám sdělí nadmořskou výšku v místě, ve kterém se zrovna nacházíte. Jaký je tedy postup? Začnete svůj sestup na náhodném, neznámém místě na úpatí hory, zkontrolujete přístroj, a pokud je Vaše nová poloha výš než poloha výchozí, tak víte, že jdete špatným směrem. Otočíte se, vyrazíte jiným směrem a proces opakujete, až dokud nedorazíte do cíle – údolí pod horou.

Analogii dokončí vysvětlení:

$$\text{Přístroj ukazující nadmořskou výšku} = \text{Cílová funkce}$$

$$\text{Ušlá vzdálenost mezi kontrolami výšky} = \text{Learning rate}$$

$$\text{Směr kroků} = \text{Záporný gradient}$$

Přeložme teď tuto analogii do jazyka matematiky.

Cílová funkce:

$$C_F(W_i) = \frac{1}{2n} \cdot \sum_{i=0}^{n} (\tilde{y}(x_i) - y)^2, \qquad (6)$$

kde y je hodnota, které systém ve zkoumaném bodě nabývá; $\tilde{y}$ je odhad hodnoty; W je měněná váha a n je počet zkoumaných bodů.

Gradient cílové funkce:

$$\frac{\partial C_F(W_i)}{\partial W_i} = \frac{1}{n} \cdot (\tilde{y}(x_i) - y)x_i \qquad (7)$$

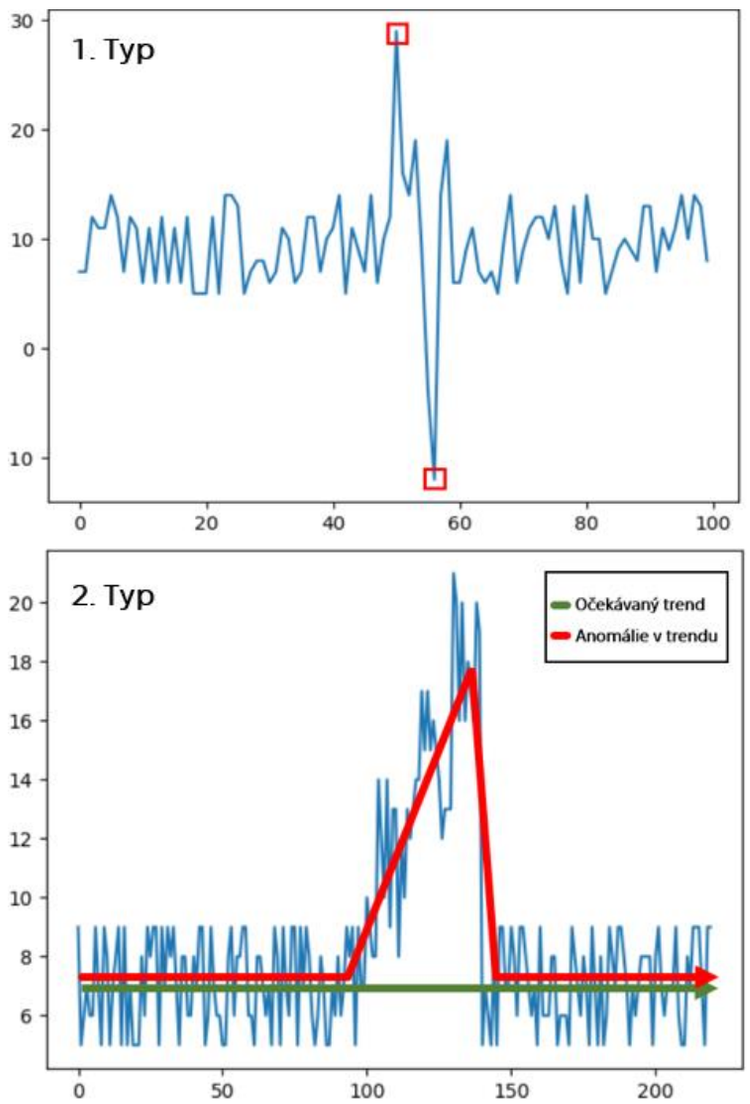
Rovnice úpravy vah má tedy tvar:

$$W_i = W_i - \frac{\mu}{n} \cdot \sum_{i=0}^{n} (\tilde{y}(x_i) - y)x_i \qquad (8)$$

2.4. APLIKACE STROJOVÉHO UČENÍ– DETEKCE ANOMÁLIÍ

Detekcí anomálií nazýváme v kontextu strojového učení proces, ve kterém se snažíme identifikovat v datech body, které nezapadají do celkového vzorce

dat prezentovaných. Anomálie se v datech mohou prezentovat mnoha způsoby. Představme si dvě z nejčastějších manifestací. První je případ, kdy datový bod je atypický v referenci k normálnímu rozložení dat ostatních. Druhým klasickým případem je neobvyklá změna normálního rozložení dat v čase. [17]



Obrázek 10 - První a druhý typ anomálie

Motivace pro identifikaci vymykajících se, a jinak chybných dat je zřejmá.

Například při měření fyzického systému sensory, se dá předpokládat občasné selhání součástek. Toto selhání způsobí v měřených datech odchylku od normy a dobře optimalizovaný algoritmus na detekci anomálií tuto

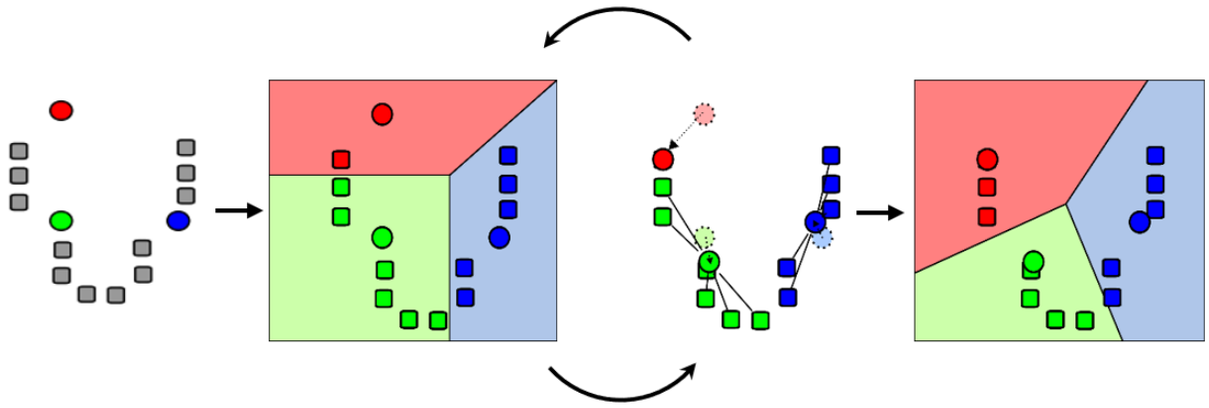
odchylku zachytí, zpracuje a oznámí. Svoji roli hraje však detekce anomálií nejen v oblasti průmyslu a sensoriky. Dalším odvětvím, které může sloužit jako dobrá ukázka užitečnosti, je například bankovnictví. Každá transakce, kterou klient provede je uložena a pomáhá tvořit obrázek o klientově zvycích a běžných útratách. Když je registrována transakce, která nezapadá do naučeného „normálu", je banka upozorněna a klient následně kontaktován pro ověření.

2.5. VYBRANÉ METODY

V této podkapitole bude čtenáři ve zkratce představeno několik vybraných metod a algoritmů. Pro algoritmy, které budou dále použity v praktické části bude vysvětlení v této podkapitole kratší, jelikož budou podrobně popsány při jejich aplikaci.

2.5.1. K – MEANS

K-Means je velice populárním algoritmem shlukové analýzy (clustrovacím algoritmem). Jednotlivé shluky (clustery) jsou definovány svými centroidy. Každý bod v prostoru je přiřazen právě jednomu shluku, a to tomu, k jehož centroidu jsou nejblíže. Písmeno **K** v názvu algoritmu připomíná, že při aplikaci K-Means musí uživatel předepsat, do kolika shluků bude sada dat rozčleněna. Tento nutný předepisovaný parametr má právě značku **K.**



Obrázek 11 - K-Means Algoritmus [18]

Na Obrázek 11 - K-Means Algoritmuslze jednoduše vidět postup shlukování pomocí K-Means. Při inicializaci je do prostoru mezi všechny body clustrované sady dat náhodně umístěno **K** centroidů. Poté je každý bod přiřazen jednomu clusteru a to právě tomu, k jehož centroidu má nejmenší euklidovskou vzdálenost. V dalším kroku jsou polohy centroidů změněny tak, aby se nacházely v těžišti spojnic všech bodů, které byly přiřazeny do jejich clusterů v prvním kroku. Kroky 2 a 3 se opakují dokud se polohy centroidů nepřestanou výrazně měnit, neboli nedojde ke konvergenci algoritmu.

Algoritmus K-Means není pro klasifikaci časových řad dobře aplikovatelný. Uživatel musí předem zadat počet shluků, na kolik se má sada dat rozdělit. Při sbírání různých druhů v čase se měnících dat K-Means selhává. I kdybychom pomocí strojového učení určili, na kolik shluků je optimální rozdělit počáteční sadu dat, tak se tento počet může v čase měnit.

2.5.2. DBSCAN

„Density Based Spacial Clustering of Applications with Noise“ je dalším z řady clusterovacích algoritmů. Přeloženo z angličtiny: „Shlukování dat na základě hustoty pro aplikace s šumem“. V případě DBSCANu se, stejně jako u K-Means, se jedná o algoritmus strojového učení „bez učitele“ (nesupervizovaný). Jak již název napovídá, DBSCAN rozděluje sadu dat na shluky podle hustoty populace v prostoru. V místech s řidší populací nemá

tendenci vytvářet shluky, nýbrž charakterizovat body, které nemají dostatečný počet sousedů, jako šum. Tato vlastnost se dá využít právě při hledání anomálií v datech. Pokud bod nezapadá do žádného z vytvořených shluků, vzniká důvodné podezření, že se jedná právě o anomálii.

DBSCAN je velice robustní a jednoduchý model. Hlavní výhodou, při aplikaci na časové řady je fakt, že uživatel nemusí při inicializaci modelu určovat, na kolik shluků se bude sada dat dělit. Počet shluků si model určí sám podle ostatních nastavených parametrů. Z těchto, a dalších důvodů bude DBSCAN jedním z algoritmů použitých v praktické části práce.

2.5.3. ARIMA

Název ARIMA modelu pochází z anglické zkratky „AutoRegressive Integrated Moving Average". Přeloženo do češtiny: „Autoregresní integrovaný klouzavý průměr". Model se skládá ze tří složek, a to ze složky autoregresní, složky klouzavého průměru a složky integrační. V kombinaci jsou schopny kvalitně analyzovat a předpovídat časové řady. Shrňme si krátce princip ARMA modelu. [19]

Autoregresní složka modelu k získání predikce využívá minulých hodnot zkoumané sady dat. Matematicky lze zapsat takto:

$$\widehat{y_t} = \beta_0 + \sum_{1}^{n} \beta_n \cdot y_{t-n} \qquad (9)$$

Kde $\widehat{y_t}$ značí predikci pro zkoumané časové okno; y_{t-n} reálné hodnoty opožděné o n časových oken. β_0 je konstanta a β_n jsou parametry opožděných hodnot, které určují velikost jejich zásahu do predikce.

Analogicky můžeme popsat i složku klouzavého průměru. Tato část modelu využívá odchylek našich minulých predikcí od korespondujících reálných hodnot. Matematicky zapíšeme:

$$\widehat{y_t} = \Phi_0 + \sum_{1}^{k} \Phi_k \cdot \epsilon_{t-k} \qquad (10)$$

Kde $\widehat{y_t}$ opět značí predikci pro zkoumané časové okno, Φ_0 je opět konstanta, Φ_k jsou opět parametry a ϵ_{t-k} je odchylka od našich předešlých predikcí. Tato odchylka většinou vzniká vnějšími, neovlivnitelnými ději.

Poslední, integrační složka modelu ARIMA se zajímá o stacionárnost zkoumané sady dat. [20] V případě, že naše zkoumaná proměnná není stacionární, je možné stacionárnosti dosáhnout pomocí diferencování. Je tedy běžnou praxí zkoumat diferencovanou proměnnou Δy_t, jednoduše získanou podle:

$$\Delta y_t = \sum_{0}^{k} (y_{t-k} - y_{t-k-1}) \qquad (11)$$

Kde k je, jako tomu bylo u složky autoregresní a složky klouzavého průměru, hodnota určující řád složky.

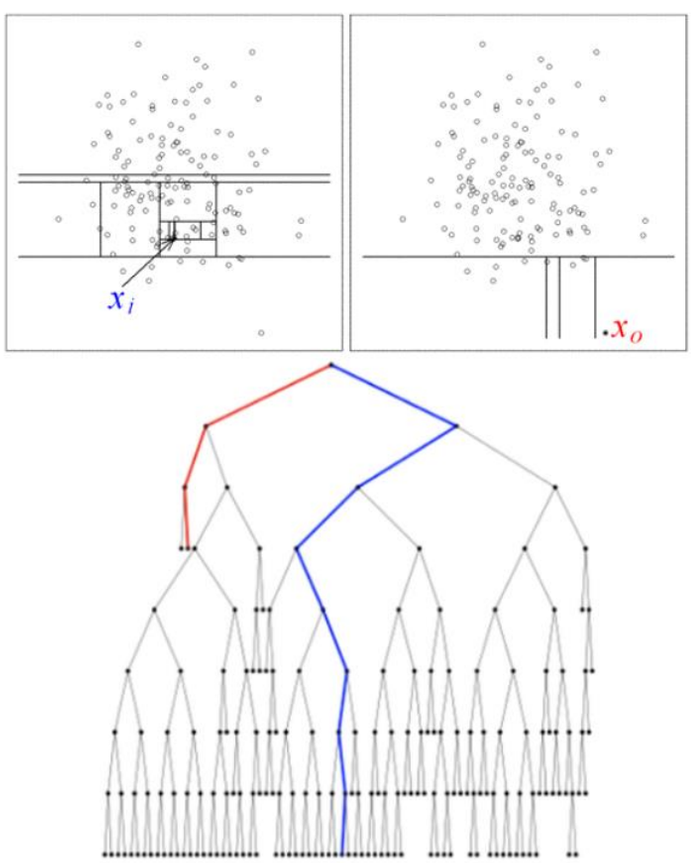
Pro vznik celkového ARIMA modelu se pak už pouze rovnice 7 a 8 jednoduše sečtou.

$$\widehat{y_t} = \beta_0 + \sum_{1}^{n} \beta_n \cdot y_{t-n} \Phi_0 + \sum_{1}^{k} \Phi_k \cdot \epsilon_{t-k} \qquad (12)$$

2.5.4. ISOLATION FOREST

„Isolation Forest“, volně přeloženo jako izolující les, je další z algoritmů hojně využívaných při detekci anomálií. Jedná se o algoritmus ze skupiny „stromových“ algoritmů a může být implementován s učitelem či bez, v praxi však hojnější využití nachází ve své nesupervizované formě [21]. Jak již název napovídá, jde o kombinaci jednotek – stromů v jeden celek – les.

Dvěma hlavními rozdíly mezi Isolation Forest a většinou ostatních metod detekce anomálií jsou fakty, že Isolation Forest neprofiluje standardní datové body, podle kterých by hledal anomálie a nepočítá žádné vzdálenosti. Anomálie identifikuje tak, že pseudonáhodně rozděluje sadu dat a staví tak obdobu známého „decision tree“. Data jsou rozdělována, dokud každé jednotlivé hodnotě není přiřazena její „větev stromu“.



Obrázek 12 - Příklad Isolation Forest, [22] [23]

Odlehlé hodnoty, na obrázku 12 vyznačeny červeně, jsou od zbytku dat odizolovány značně „výše“, než hodnoty pro sadu dat standardní. Tato délka větve pak hraje hlavní roli při skórování anomálie. Každému bodu je, jak je v algoritmech detekce anomálií klasické, totiž přiřazeno tzv. „anomální skóre“ a to podle:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \qquad (13)$$

kde $s(x,n)$ je anomální skóre, $h(x)$ je délka větve, $E(h(x))$ je střední hodnota $h(x)$ ze celého lesa – skupiny stromů a $c(n)$ je komplexně počítaná konstanta závislá na n – počtu bodů ve zkoumané sadě dat [23].

Na obrázku 12 lze vidět fungování tohoto skóre. U červeně vyznačené anomálie bylo zapotřebí značně méněkrát sadu dat rozdělit, abychom izolovali jediný bod. Naopak izolovat bod modrý, který je svými vlastnostmi pro zadanou sadu dat obvyklý, se rozdělit povedlo až po mnoha „rozvětveních“.

Isolation Forest má mnoho použití a je to z pohledu výpočetní a časové náročnosti velmi zdvořilý algoritmus. Více k němu bude popsáno v praktické části.

2.5.5. STATISTICKÉ PROFILOVÁNÍ

Pravděpodobně nejzákladnějším, a často prvním voleným, přístupem k detekci anomálií je jednoduchý přístup statistického profilování. Analogicky se známou metodou z oblasti technologií výroby Six Sigma (česky „šest sigma“), náš statistický přístup označuje anomální body s využitím standardní odchylky.

Princip této metody je velice prostý jak po teoretické stránce, tak při implementaci. Jednoduše se u zkoumané sady dat spočte její střední hodnota, variace a z ní standardní odchylka. Střední hodnotu spočteme podle:

$$\mu = \frac{\sum_{i=1}^{i} x_i}{n} = E(x) \qquad (14)$$

Značení E(x) pro střední hodnotu sady dat vychází z anglického „Expected value" neboli očekávaná hodnota. Je uvedena, jelikož se zápis této veličiny liší v české a zahraniční literatuře.

Varianci, neboli rozptyl, dále spočítáme jako součet rozdílů jednotlivých hodnot od hodnoty očekávané, podělený počtem hodnot podle:

$$var(x) = \frac{1}{n}\sum_{i=1}^{n}[x_i - E(x)]^2 = \sigma^2 \qquad (15)$$

A poté už jednoduše odmocníme a dostáváme směrodatnou odchylku podle:

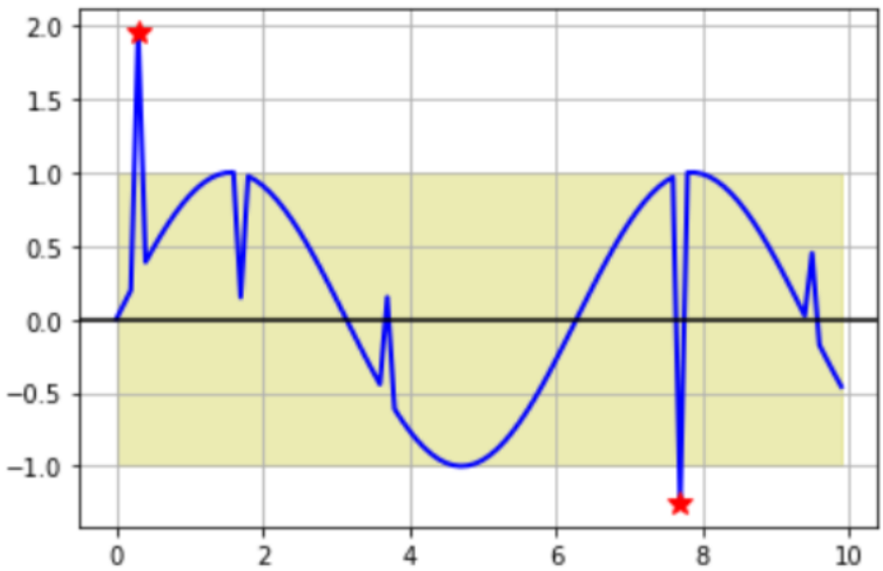
$$\sigma = \sqrt{var(x)} \qquad (16)$$

Pomocí těchto hodnot můžeme vytvořit pomyslné, s každou další hodnotou se měnící, pásmo, do kterého musí hodnoty spadat. Hodnoty, které se ocitají mimo toto pásmo jsou označeny za anomálie, viz Obrázek 13.

Hodnoty mezních přímek pásma matematicky zapíšeme a spočítáme podle:

$$y_{krit._{1,2}} = \mathrm{E}(x) \pm k \cdot \sigma \qquad (17)$$

kde $y_{krit._{1,2}}$ jsou spodní a horní hodnota pásma a k je libovolná volená konstanta. Dopad volby konstanty k je zřejmý. Určuje, kolik směrodatných odchylek v jednom směru je pásmo široké.



Obrázek 13 - Příklad přístupu statistického profilování k detekci anomálií [24]

Přístup statistického profilování bude použit v praktické části.

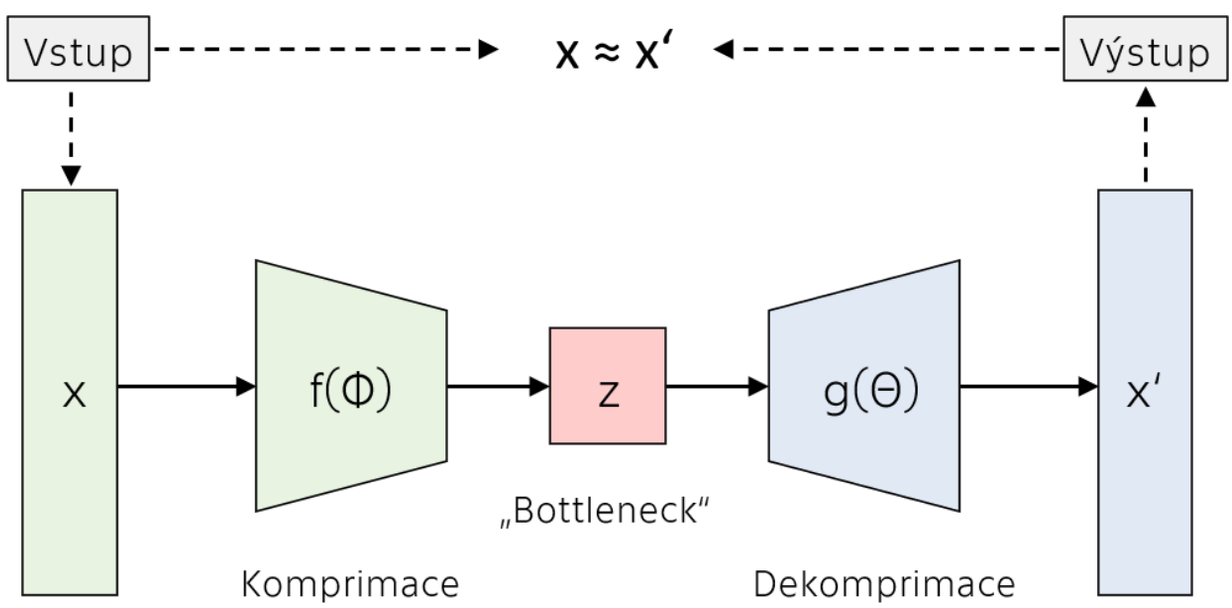
2.5.6. LSTM AUTOENKODÉR

Poslední metodou detekce anomálií, kterou si v této části práce probereme jeden z typů hlubokých neuronových sítí, tzv. LSTM-Autoenkodér. Autoenkodéry obecně nacházejí široké využití napříč obory. Jsou využívány například pro redukci dimenzionality, snížení šumu v obraze, rozpoznání objektů v obraze, a v neposlední řadě právě pro detekci anomálií.

V případě detekce anomálií v časových řadách se používá, pro tento účel uzpůsobený, LSTM-autoenkodér. Zkratka LSTM značí „Long Short-Term Memory", volně přeloženo jako „Dlouho-krátko dobá paměť". Detaily fungování tohoto specifického typu autoenkodéru jsou nad rámec této práce a nepovažuji je za kritické. Dále bude vysvětlen obecný princip fungování autoenkodéru a způsob jeho využití v detekci anomálií.

Architektura autoenkodéru má 4 hlavní komponenty, se kterými by čtenář měl být seznámen [25], [26].

1. Enkodér: Ve vrstvách enkodéru se model učí efektivní komprese rozměru vstupního vektoru tak, aby byly zachovány všechny důležité charakteristiky.
2. Bottleneck: Nejužší vrstva modelu. Obsahuje komprimovanou reprezentaci vstupního vektoru. V této vrstvě se nachází nejmenší možná reprezentace vstupních dat.
3. Dekodér: Vrstvy dekodéru dostávají vstup z „bottlenecku". Snaží se naučit, jak rekonstruovat data, tak aby se, co možná nejpřesněji, podobala vstupnímu vektoru.
4. Ztrátová funkce: Forma cílové funkce. V tomto případě použita k měření přesnosti dekodéru. Čím nižší hodnota ztrátové funkce, tím lépe se shoduje vstup x a výstup x'.



Obrázek 14 - Vizualizace Autoenkodéru, překresleno z [26]

Použití autoenkodéru k detekci anomálií funguje na základě neschopnosti rekonstrukce anomálních vstupních dat ve fázi dekomprimace. Autoenkodér je učen, jak komprimovat a dekomprimovat standardní, běžná data. Když se ve vstupních datech vyskytují anomálie, dekodér není schopen věrohodně rekonstruovat vstupní vektor a tím pak narůstá hodnota ztrátové funkce. Při

překročení nastavitelné hranice hodnoty ztrátové funkce je vstupní vektor označen jako anomální.

LSTM autoenkodér je velice silným hráčem na poli algoritmů detekce anomálií časových řad. Jeho optimální použití, a to primárně optimální použití v reálném čase daleko přesahuje rozsah této diplomové práce. Bylo by možné jeho silně sub-optimální verzi implementovat. Pro identifikaci anomálií v reálném čase by však tento sub-optimální algoritmus byl bez doprovodu specificky konfigurovaného výpočetního hardwaru téměř nepoužitelný. Z tohoto, a dalších důvodů je tento algoritmus zmíněn pouze v teoretické části práce.

3. PRAKTICKÁ ČÁST

Upozornění

Praktická aplikace témat probíraných v teoretické části práce není jednoduchá. Primárním problémem je nedostatek použitelných reálných či kvalitních dat. I přes spolupráci se společností MySCADA nebylo možné obstarat data obsahující anomálie ze skutečné výroby, a to ani pod příslibem anonymizace. Společnosti tato data veřejně neposkytují z několika prostých důvodů. Jednak může jít o ohrožení firemního know-how, dále k tomu nejsou firmy nijak motivovány, zveřejnění těchto dat jim nepřináší žádný užitek. Hlavním problémem je dle mého názoru však zachování image společnosti. Pokud nastane úspěšný kybernetický útok na výrobní společnost, je velmi nepravděpodobné, že by se tím postižená společnost chtěla veřejně chlubit. Logickým přístupem je v takové situaci snaha o zamlčení chyb a poučení se z omylů.

3.1. KONCEPT PRAKTICKÉ ČÁSTI

Kvůli již zmíněnému problému s nedostatkem dat jsem se rozhodl v praktické části ukázat přístup k detekci anomálií v částech, a na více typech dat, která se v industriálních řídících systémech objevují. V první části budu na stažených datech ze senzorů z Intel Berkley Research Lab [27] ilustrovat metodiky detekce anomálií, porovnávat jejich účinnost a použitelnost. Dále díky poznatkům navrhnu kombinaci těchto metodik, která by optimálně měla odhalovat všechny anomálie, aniž by vykazovala časté falešně pozitivní indikace anomálií.

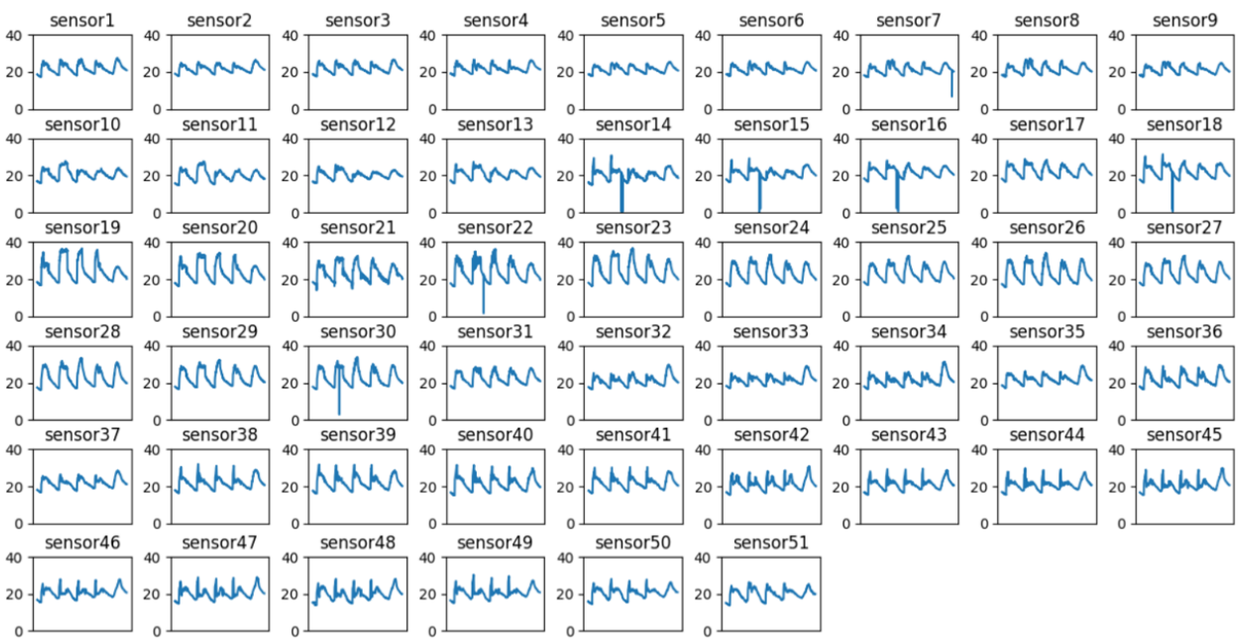
V druhé části aplikuji zmíněnou kombinaci na mnou simulovaná data, která budu v reálném čase posílat skrze, v praxi běžný protokol pro komunikaci IIoT zařízení, OPC UA.

3.2. ČÁST PRVNÍ - TESTOVÁNÍ MODELŮ

Metody detekce anomálií probrané v kapitole 4, budou aplikovány na staženou sadu dat. Jednotlivé metody budou porovnány, vyzdviženy jejich výhody a vytčeny jejich nevýhody.

3.2.1. POUŽITÁ DATA

Stažená data obsahují časové řady z 51 sensorů, viz obrázek 15. Většina časových řad na první pohled neobsahuje anomálie. Můžeme si však všimnout, například u sensorů 14-16, že některé anomálie obsahují. Každá časová řada obsahuje 14400 odečtů sensoru a u všech se hodnoty odečtů pohybují ve zhruba stejných hranicích 15 – 35.



Obrázek 15 - Data z Intel Berkley Research Lab

Předzpracování těchto dat je velmi přímočaré. Na první pohled lze vidět že data jsou stacionární a nechybí v nich žádné instance hodnot. Všechny sensory se pohybují zhruba ve stejných hodnotách, není tudíž třeba hodnoty škálovat či jinak upravovat.

Každá časová řada z této sady bude podrobena každému z probraných algoritmů, bude měřena časová náročnost, a následně bude vizuálně zhodnocen výsledek.

3.2.2. POUŽITÝ SOFTWARE

Metody budou implementovány pomocí programovacího jazyka Python3. Dále jsou uvedeny knihovny a balíčky použity při implementaci.

Visual Studio Code – Editor zdrojového kódu

NumPy – knihovna poskytující infrastrukturu pro práci s vektory, maticemi a obecně více rozměrnými poli. Dále nabízí řadu užitečných matematických funkcí jako například diskrétní Fourierovu transformaci a základy lineární algebry.

Matplotlib – knihovna používaná k vykreslování dat. Umožňuje jednoduše produkovat složité grafy. Synergie s NumPy a jinými knihovnami.

Pandas – open-source knihovna pro zjednodušení práce s daty a časovými řadami.

scikit-learn – knihovna obsahuje řadu algoritmů využívaných ve strojovém učení.

Statsmodels – balíček poskytuje přístup k řadě užitečných statistických funkcí. Umožňuje jednoduše implementovat statistické algoritmy k úpravě dat, predikci trendů a další.

timeit – Malý balíček umožňující měřit časovou náročnost kódu.

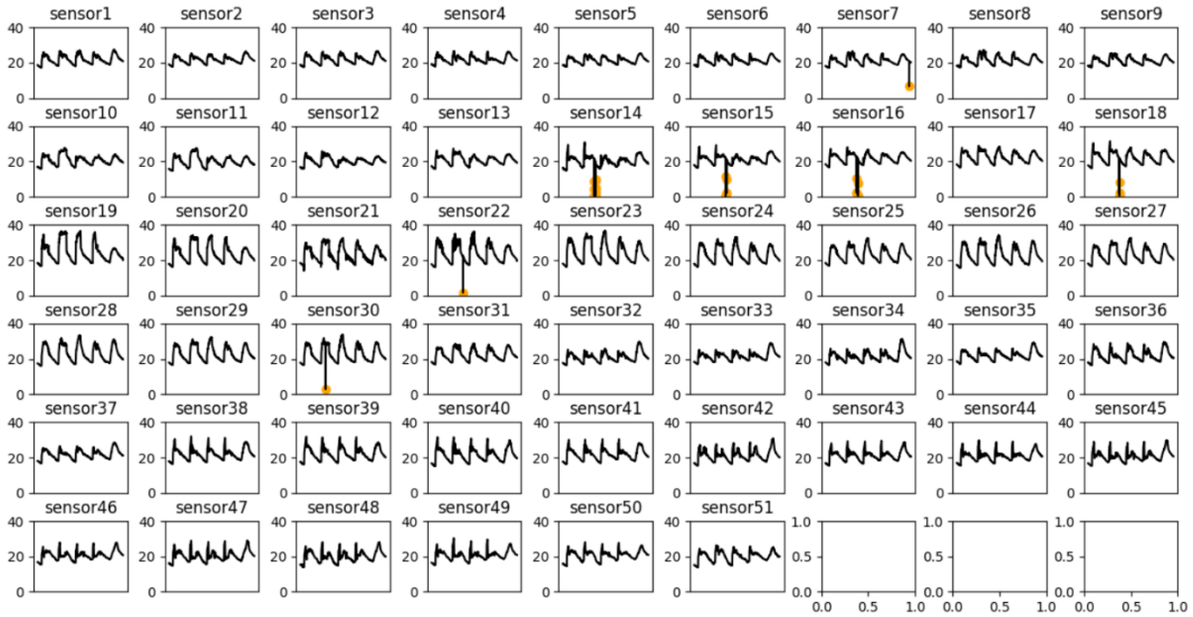
3.2.3. SROVNÁNÍ METOD

V této podkapitole budou ukázány nastavitelné a vybrané parametry, případně dovysvětleny detaily fungování, které nebyly pokryty v podkapitole 4.6.

Statistické profilování

Jediným nastavovaným parametrem ve funkci statistického profilováníje počet standardních odchylek, který určuje šířku pásma běžných hodnot. V rovnici (17) to je tedy právě konstanta k. V našem případě jsem po několika testech zvolil hodnotu **k = 4**.

Na Obrázek 16 můžeme vidět, že tento přístup bezpečně identifikuje všechny anomálie, které se vyskytují v naší sadě časových řad. Dále bohužel tento přístup může identifikovat i jiná „podezřelá místa“, která se nezdají být anomáliemi. Tato indikace falešných anomálií se dá řešit rozšířením pásma – zvýšením konstanty k. Došlo by však ke snížení citlivosti algoritmu. V případě detekce anomálií na těchto časových řadách to není problém, při použití v jiných instancích by však necitlivý algoritmus nemusel vůbec identifikovat anomální hodnotu, což je horší chybou než falešně pozitivní označení.



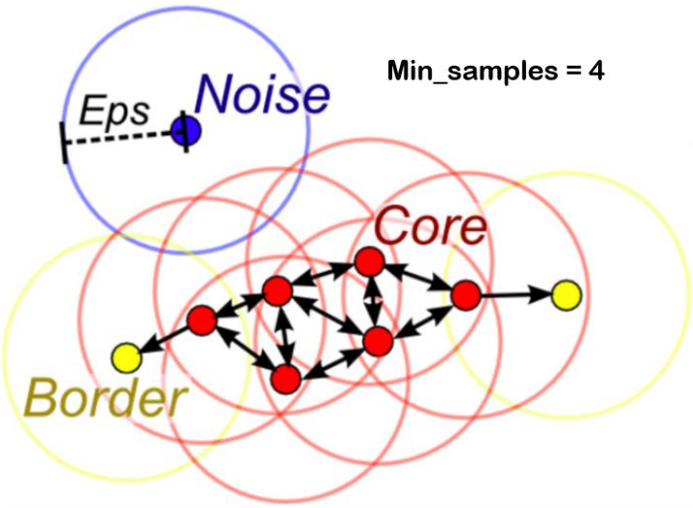
Obrázek 16 – Výsledek přístupu statistickým profilováním

Dále byl výkon algoritmu měřen v závislosti na jeho časových požadavcích. Průměrná doba, za kterou tento algoritmus identifikoval anomálie v jedné časové řadě je 0,322s.

DBSCAN

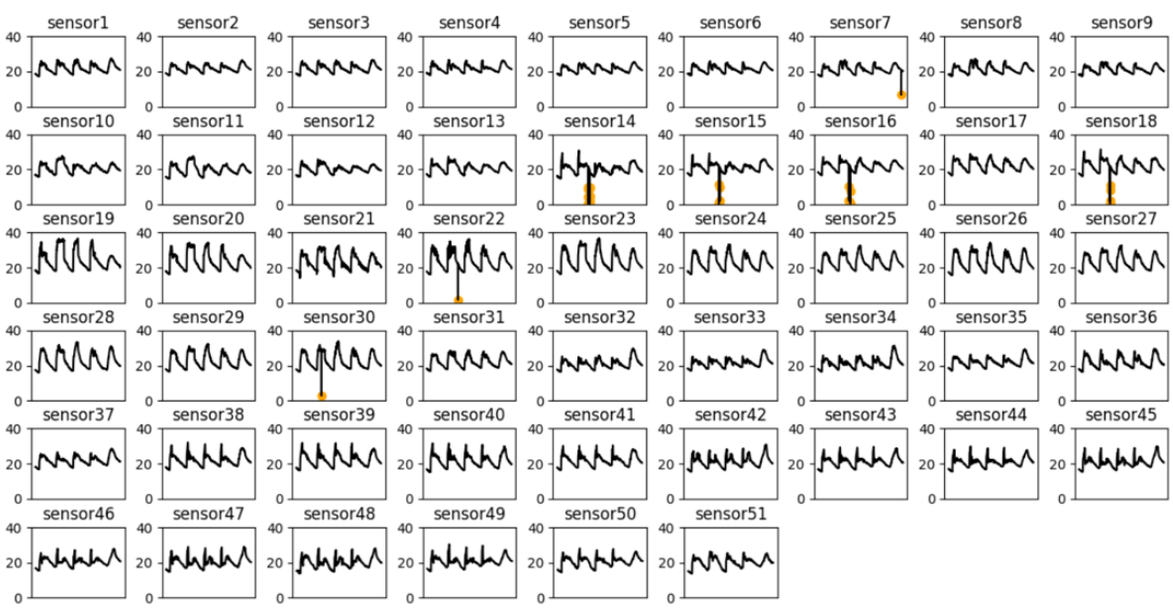
Algoritmus DBSCAN je implementován z knihovny scikit-learn. Nastavitelných parametrů je mnoho, hlavními jsou ale dva [28]. Prvním z nich je **eps** (zkratka řeckého písmene epsilon). Ten určuje maximální vzdálenost, do které jsou dva body považovány za „sousedské". Viz Obrázek 17 – . Druhým z hlavních parametrů je **min_samples** (někdy také označováno „MinPts"). Určuje minimální počet sousedních bodů, aby byly označeny jako jeden shluk.

V našem případě, jsem po několika testovacích iteracích dospěl k hodnotám **eps = 0.07**, **min_samples = 10**.



Obrázek 17 – Základní parametry DBSCAN

Na Obrázek 18 - Výsledek algoritmu DBSCAN můžeme vidět, jak efektivně DBSCAN funguje. Bez jakýchkoliv falešně pozitivních indikací odhalil všechny očividné anomálie.



Obrázek 18 - Výsledek algoritmu DBSCAN

Dále byla změřena časová náročnost DBSCAN algoritmu. Průměrná doba, za kterou algoritmus identifikoval anomálie v jedné časové řadě je 1,574s.

Isolation Forest

Isolation Forest je také implementován ve své formě z knihovny scikit-learn. Stejně jako DBSCAN, a většina pokročilých algoritmů, má mnoho nastavitelných parametrů. Primární jsou v tomto případě 3. Prvním z nich je **n_estimators**. Tento parametr určuje počet stromů, ze kterých se skládá celý isolation forest. Pro výpočet optimální hodnoty existují algoritmy, pro naše účely jsem však metodou „pokus-omyl" konvergoval k hodnotě **n_estimators** = 2. V našem případě se tedy nejedná tolik o les, spíše o pár stromů. Vyšší hodnoty tohoto parametru dávali shodné výsledky, časová náročnost však lineárně s každým dalším stromem stoupala.

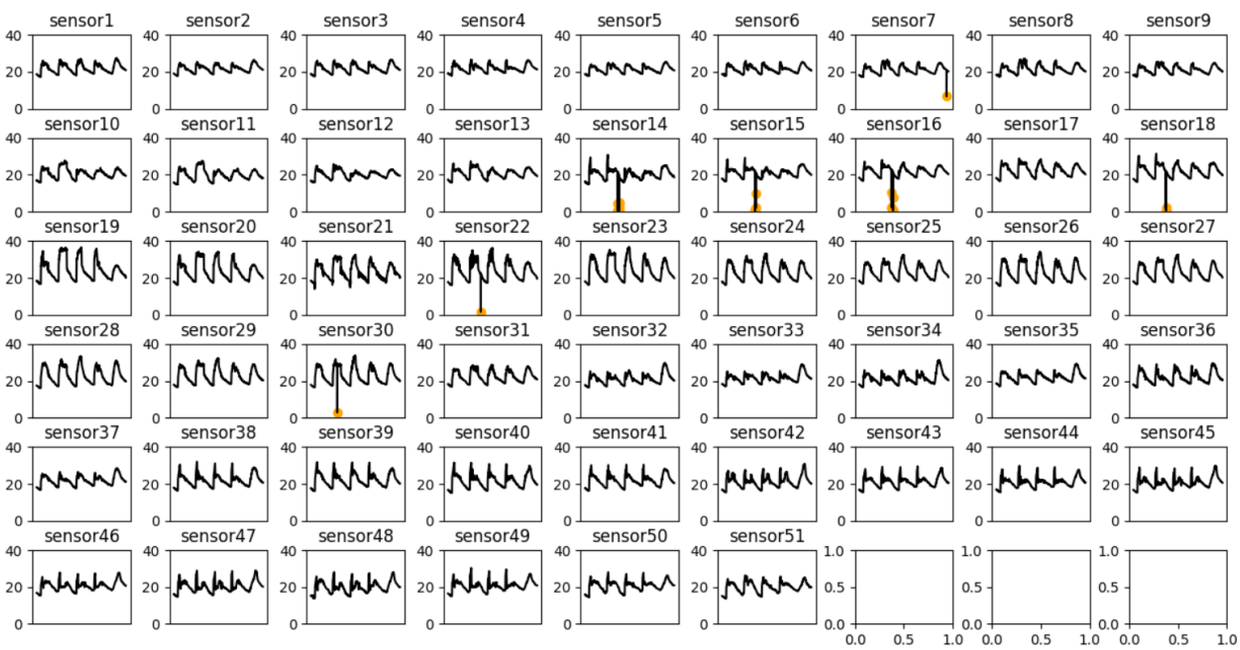
Druhým parametrem je **max_samples**. Tento parametr určuje rozdělení celé sady dat na část trénovací a část testovací. Po několika pokusech jsem se rozhodl nastavit **max_samples = 0.6**, tedy 60 % dat je použito při trénování.

Posledním, a pravděpodobně nejdůležitějším parametrem je tzv. **contamination**, neboli „kontaminace". Označuje kolik procent ze sady dat očekáváme anomálních. V tomto parametru lze najít skvělou synergii s metodou statistického profilování, či dalšími metodami. Pokud necháme sadu dat otestovat nejdříve jiným algoritmem, můžeme tento odhad dostávat průběžně a v reálném čase, jako výstup z něj. V tomto případě tedy nastavuji **contamination = contamination_statistical**, kde „contamination_statistical" je výstup z algoritmu statistického profilování, který se řídí podle:

$$\mathbf{contamination_statistical} = \frac{n_{anomálií}}{n_{celku}} \qquad (18)$$

Dalšími nastavenými parametry, které jsou méně podstatné a nebudou vysvětleny jsou: **max_features = 1**; **n_jobs = -1**; **random_state = 1**; **warm_start = True**

Na Obrázek 19 – Výsledek Isolation Forestvidíme, jak Isolation Forest s těmito parametry kvalitně identifikuje anomálie v časových řadách. Je schopen bez problému rozpoznat všechna problematická místa bez indikace falešně pozitivních anomálií.



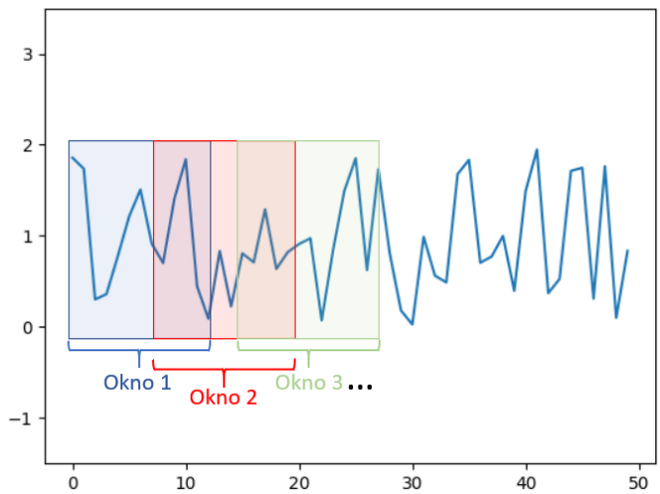
Obrázek 19 – Výsledek Isolation Forest

Identifikace anomálií v jedné časové řadě průměrně trvala takto nastavenému Isolation Forest algoritmu pouhých 0,513s.

ARIMA

Model ARIMA implementuji z knihovny Statsmodels. Aplikace tohoto modelu na sadu dat, která není generována v reálném čase, vyžaduje použití techniky zvané Sliding Window (přeloženo z angličtiny jako „klouzavé okno“).

Ve zkratce vysvětleno na Obrázek 20 - Grafická reprezentace Sliding Window, Sliding Window je přístup, kdy vstupní data neanalyzujeme všechna najednou, ale postupně po skupinách zvaných „okna“. Každé další okno nezačíná tam, kde předchozí okno skončilo, nýbrž pouze o n datových bodů dál, velmi často (a v našem případě) pouze o jeden.



Obrázek 20 - Grafická reprezentace Sliding Window

Model ARIMA není inherentně algoritmus detekce anomálií, využitím jeho schopnosti predikce časových řad, ho však můžeme tak využít. Pro každé okno můžeme nechat ARIMA odhadnout, jak by se měla časová řada chovat dál. Dále můžeme změřit velikost chyby, které se model při predikci dopustil. Pokud chyba překračuje nastavenou hranici, je bod klasifikován jako anomální.

Hlavními nastavitelnými parametry v modelu ARIMA jsou tyto 3:

p – počet minulých datových bodů, jejichž hodnoty brát v potaz. V rovnici (9) značeno jako n.

d – kolikrát je řada diferencována. V rovnici (11) jako k.

q – počet minulých datových bodů, ze kterých počítat klouzavý průměr. V rovnici (10) jako k.

Dalším důležitým parametrem při implementaci ARIMA je jednoznačně velikost (délka) sliding window. Měření výsledků a časové náročnosti algoritmu ARIMA by tedy bylo silně zatíženo kvalitou optimalizace našeho Sliding Window. Nebude proto experiment pro ARIMA proveden v této části. Model bude s ostatními srovnán až v části druhé.

Shrnutí

Algoritmus	Identifikuje anomálie?	Falešně pozitivní indikace?	Časová náročnost	Složitost implementace	Parametry
Statistical Profiling	✓	~	Nízká	Nízká	k = 4
Isolation Forest	✓	✗	Nízká	Střední	n_estimators = 2 max_samples = 0.6 contamination
DBSCAN	✓	✗	Střední	Střední	eps = 0.07 min_samples = 10
ARIMA	~	~	Vysoká	Vysoká	p,d,q l_window

Tabulka 1 – porovnání probraných algoritmů

3.3. ČÁST DRUHÁ – ČÁST HLAVNÍ

Druhá část praktické implementace by se dala nazvat částí hlavní. Naše poznatky z teorie a z implementace modelů do praxe, budou nyní použity k detekci anomálií komunikace mezi serverem a clientem přes protokol OPC UA v reálném čase.

3.3.1. POUŽITÝ SOFTWARE

V této části budou metody také implementovány pomocí programovacího jazyka Python3. Knihovny, probrané v testovací části budou použity stejné, přidá se k nim pouze pár dalších.

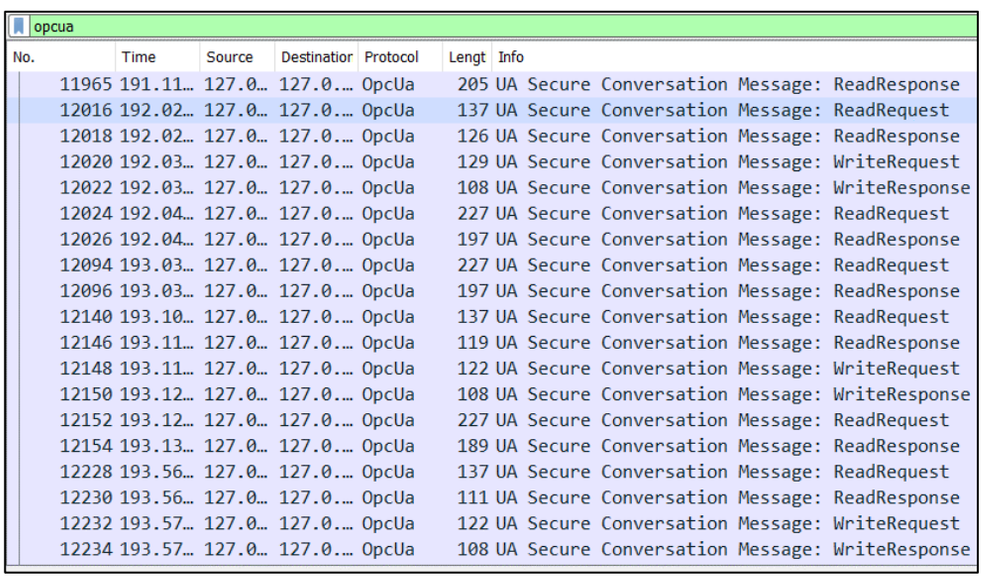
opcua

Knihovna se stejným názvem jako protokol, který zprostředkovává, by se dala nazvat páteří této práce. Umožňuje vytvořit dvojici server – client a posílat skrze toto spojení packety informací. Dále umožňuje definovat

proměnné, psát interní logiku serveru či nastavovat bezpečnostní politiku serveru.

WireShark/TShark/PyShark

Pro účely monitorování komunikace mezi párem server-client použijeme software pod názvem WireShark. Rozhraní softwaru na Obrázek 21.


opcua

No.	Time	Source	Destination	Protocol	Lengt	Info
11965	191.11…	127.0…	127.0.…	OpcUa	205	UA Secure Conversation Message: ReadResponse
12016	192.02…	127.0…	127.0.…	OpcUa	137	UA Secure Conversation Message: ReadRequest
12018	192.02…	127.0…	127.0.…	OpcUa	126	UA Secure Conversation Message: ReadResponse
12020	192.03…	127.0…	127.0.…	OpcUa	129	UA Secure Conversation Message: WriteRequest
12022	192.03…	127.0…	127.0.…	OpcUa	108	UA Secure Conversation Message: WriteResponse
12024	192.04…	127.0…	127.0.…	OpcUa	227	UA Secure Conversation Message: ReadRequest
12026	192.04…	127.0…	127.0.…	OpcUa	197	UA Secure Conversation Message: ReadResponse
12094	193.03…	127.0…	127.0.…	OpcUa	227	UA Secure Conversation Message: ReadRequest
12096	193.03…	127.0…	127.0.…	OpcUa	197	UA Secure Conversation Message: ReadResponse
12140	193.10…	127.0…	127.0.…	OpcUa	137	UA Secure Conversation Message: ReadRequest
12146	193.11…	127.0…	127.0.…	OpcUa	119	UA Secure Conversation Message: ReadResponse
12148	193.11…	127.0…	127.0.…	OpcUa	122	UA Secure Conversation Message: WriteRequest
12150	193.12…	127.0…	127.0.…	OpcUa	108	UA Secure Conversation Message: WriteResponse
12152	193.12…	127.0…	127.0.…	OpcUa	227	UA Secure Conversation Message: ReadRequest
12154	193.13…	127.0…	127.0.…	OpcUa	189	UA Secure Conversation Message: ReadResponse
12228	193.56…	127.0…	127.0.…	OpcUa	137	UA Secure Conversation Message: ReadRequest
12230	193.56…	127.0…	127.0.…	OpcUa	111	UA Secure Conversation Message: ReadResponse
12232	193.57…	127.0…	127.0.…	OpcUa	122	UA Secure Conversation Message: WriteRequest
12234	193.57…	127.0…	127.0.…	OpcUa	108	UA Secure Conversation Message: WriteResponse

Obrázek 21

Řádky reprezentují jednotlivé packety, proudící skrze komunikační protokol OPC UA. V každém packetu jsou zašifrované informace, které packet přenáší, stejně jako informace o velikosti packetu, typu packetu či informace o čase, ve kterém packet dorazil. Tyto informace může člověk ručně procházet a číst, my však máme snahu o automatizaci tohoto procesu. V tomto nám pomáhá TShark, což je WireShark spustitelný a ovladatelný přímo z Windows terminálu. Přesněji použijeme PyShark, což je pythonovský wrapper pro TShark. PyShark disponuje řadou užitečných funkcí, použitelných při dešifrování packetů a extrahovaní obecných informací o packetu.

Time

Z názvu by mělo být jasné, o co se tato pythonovská knihovna stará. Umožňuje uživateli využívat časování ve svých programech. Dává přístup k užitečným funkcím spojených s časem.

myDESIGNER8

Vývojové prostředí pro tvorbu vizualizací IIoT projektů od české společnosti mySCADA. Pomáhá jednoduše tvořit HMI – Human-Machine Interface (přeloženo z angličtiny jako „rozhraní člověk-stroj"). Nabízí také automatické spojení se serverem skrze protokol OPC UA, tím zjednodušuje nastavování tagů a obecně konfiguraci HMI. Software detekuje všechny proměnné, definované na straně serveru a automaticky jim přiřazuje patřičné tagy. Dále lze jednotlivé proměnné – jednotlivé tagy přiřadit komponentům v HMI. Uživatel tak dokáže svázat serverovou proměnnou s vizualizací, například tlačítkem, díky kterému může třeba měnit hodnotu proměnné.

3.3.2. TESTOVACÍ PŘÍPAD

Pro jednodušší pochopení problematiky si definujme případ z reálného světa, jehož data se pokusíme simulovat v reálném čase. Inspirujme se v podkapitole 1.2.3. a případu útoku na Iránskou továrnu na obohacený uran. Pozorný čtenář si jistě pamatuje že šlo o útok, který měnil otáčky industriálních odstředivek a tím znehodnocoval jejich produkt a je samotné.

Pro náš testovací případ si tedy představme jednoduchý ovladač takové industriální odstředivky, která má vnitřní nastavení na 1500 otáček/min.

K těmto otáčkám generujme nejistotu ±5 otáček/min jako simulaci šumu pro testování citlivosti algoritmů.

Cílem experimentu bude měnit hodnotu otáček a sledovat, zda náš algoritmus dokáže spolehlivě odhalit neočekávané změny jako anomální. Tyto otáčky pomyslné odstředivky jsou dále označovány jako parametr „Packet_Value".

3.3.3. SERVER – CLIENT

Jak již bylo zmíněno, server spouštíme skrze programovací jazyk Python3 a přidruženou knihovnu opcua. Na serverové straně definujeme nezapisovatelné proměnné, které bude client – HMI, zobrazovat. Dále definujeme zapisovatelné proměnné, kterými můžeme podávat z HMI vstup do logiky serveru.

Bezpečnostní certifikát serveru vypněme, abychom zjednodušili čtení obsahu packetů. V reálném případě by samozřejmě obsah packetů byl silně zašifrován, a šel by číst pouze uživateli, kteří disponují bezpečnostním klíčem.

Server poběží pouze lokálně na počítači. Pro přístup použijeme tedy adresu 127.0.0.1, známou také jako „localhost", a nastavitelný port 4841. Clientem je v našem případě HMI s ovládacími tlačítky vyobrazené na Obrázek 22 -HMI. Pro sjednocení čtení a zápisu dat z detekčního softwaru jsou nastaveny vzorkovací frekvence obou částí softwaru na 4 Hz, fázově posunuto. Když by totiž detekční software chtěl zapisovat své poznatky do „.csv" složky, zatímco ji má otevřenou server, došlo by k pádům jednoho nebo obou částí softwaru.

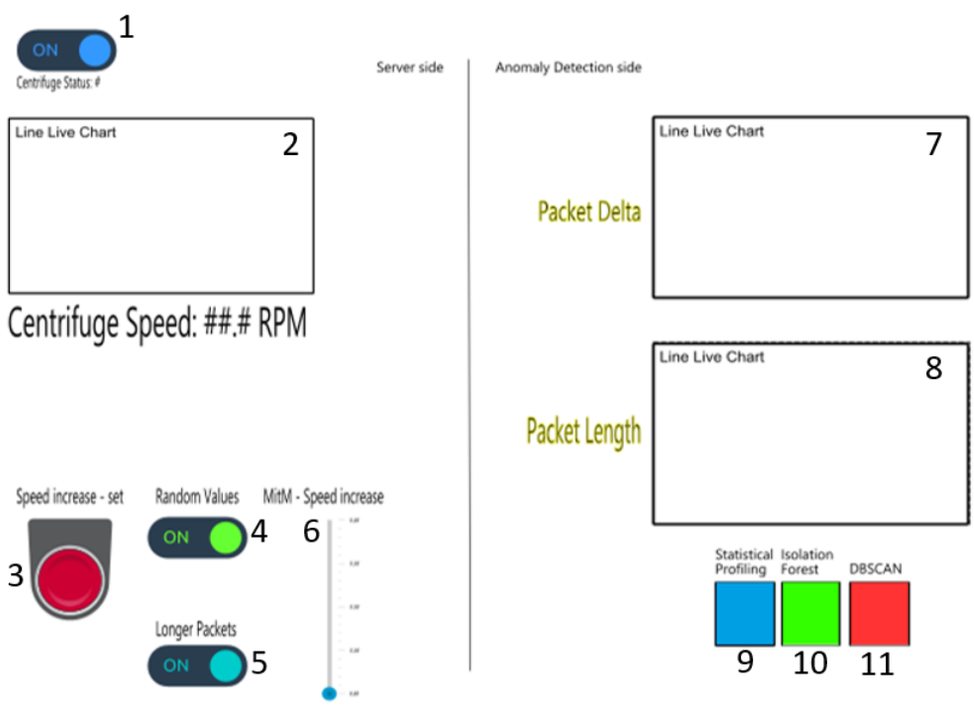
Popišme si jednotlivé části HMI na Obrázek 22 -HMISamotné HMI je vizuálně rozděleno na dvě části, aby bylo na první pohled jasné, o které informace se stará sám server a o které se stará detekční algoritmus.

Strana serveru

1- Ovládání zap./vyp. pomyslné odstředivky; 2- Graf zobrazující rychlost odstředivky (pod ním i číselně); 3- Tlačítko na zvýšení rychlosti o danou hodnotu; 4- Vypínač na generátor náhodných hodnot. Při zapnutí se rychlost v každém okamžiku značně mění; 5 – Přepínač velikosti packetů. Při zapnutí obsahují packety značně více bajtů; 6 – jezdec upravující rychlost odstředivky.

Strana softwaru detekce anomálií

7 – packet__delta zobrazováno v reálném čase; 8 – Packet__Length zobrazováno v reálném čase; 9,10,11 – indikace jednotlivých detekčních algoritmů, když se kontrolka rozsvítí, algoritmus detekuje anomálii.



Obrázek 22 -HMI

3.3.4. VYHODNOCOVANÉ ATRIBUTY

Rozeberme si data, která budeme analyzovat pro potenciální anomálie. Při výběru takzvaných „features“ u kterých budeme detekovat anomálie se

odrazme od kapitoly 1.2. Pokud budeme brát v potaz typy útoků v této kapitole probrané, pak se musíme bránit proti útokům typu DDoS a Man-in-the-Middle. N-day exploit vynecháváme, protože jak bylo v kapitole 1.2.3. zmíněno, obrana proti tomuto typu útoku nevězí v analýze packetů.

Hledejme tedy vlastnosti packetů, které by mohly být ovlivněny při DDoS a Man-in-the-Middle útocích. Z poznatků z kapitol probírajících problematiku DDoS útoku je jasné, že v případě pokusu o shození průmyslové sítě bychom detekovali silně zvýšenou hladinu proudících packetů. U některých variant DDoS útoku, jmenovitě tzv. „Ping of Death" bychom také zaznamenali extrémní nárůst v počtu bajtů, které každý packet přenáší. Jeden feature je tedy jasný, v případě DDoS útoku bychom měli monitorovat **velikost každého packetu**. Tento vstupní vektor nazvěme **Packet_Length**. Obyčejný packet při běžné komunikaci na mnou vytvořeném serveru obsahuje mezi 120 a 240 bajty. Při útoku typu „ping of death" může počet bajtů stoupnout až na mezní hranici protokolu TCP/IP, a to 65535 bajtů. Velikost atribut Packet_Length je v čase téměř konstantní.

V případě MitM útoku, jak bylo zmíněno v jemu věnované podkapitole, musí hacker každý packet rozbalit, přečíst či změnit jeho „payload", zase zabalit a poslat dál. Toto resultuje v prodloužení cesty každého packetu. Druhý vstupní feature získáme právě z tohoto poznatku. Každý packet ve svých metadatech obsahuje čas, kdy byl packet obdržen. V případě MitM útoku můžeme tedy zkoumat rozdíl časů, ve kterých jsme obdrželi každý packet od packetu předchozího. Tento feature nazvěme **Packet_Delta**.

$$Packet\_Delta = \Delta t = t_{packet_n} - t_{packet_{n-1}} \qquad (19)$$

Feature Packet_Delta je užitečný také při detekci útoku typu DDoS, monitoruje totiž nepřímo i počet proudících packetů. Laicky řečeno, pokud se hodnota Packet_Delta skokově sníží, znamená to větší proud packetů za

sekundu, může se jednat o DDoS útok. Naopak pokud se hodnota Packet__Delta zvýší, může se jednat o útok typu MitM. Atribut Packet__Delta je vysoce závislý na rychlosti hardwaru, na kterém běží server. Vzhledem k proměnlivému výkonu mého laptopu se Packet__Delta v čase nepředvídatelně mění a způsobuje falešně pozitivní indikace u některých metod.

Posledním vstupním vektorem, který budeme používat je nepřekvapivě obsah packetu. U každého packetu, který nastavuje hodnotu otáček naší pomyslné odstředivky, si packet rozbalíme, a zkontrolujeme, zda hodnota, kterou se packet snaží nastavit odpovídá normálu. Feature **Packet__Value** je tedy užitečný v prevenci MitM útoků. V Tabulka 2 přehledně které vstupní vektory detekují jaký typ útoku.

Feature	DDoS ovlivňuje	MitM ovlivňuje
Packet__Length	✓	✗
Packet__Delta	✓	✓
Packet__Value	✗	✓

Tabulka 2

3.3.5. JEDNOTLIVÉ ALGORITMY A JEJICH VÝSLEDNÁ KOMBINACE

Po prozkoumání mnoha algoritmů zůstaly pouze 4, které se ucházejí o místo ve výsledném softwaru. Jsou to metoda statistického profilování, dále Isolation Forest, pak ARIMA a v neposlední řadě DBSCAN. Kritéria výběru pro aplikaci ve výsledném softwaru jsou mnohá a zdaleka nebudou všechna popsána. Proberme si ta hlavní. Samozřejmostí je identifikace anomálií.

V případě, že zapneme libovolný typ simulace útoku, tak by se v ideálním případě měly rozsvítit indikátory anomálií všech algoritmů. Druhým kritériem je minimum falešných indikací anomálií, tzn. pokud není zapnutá žádná simulace útoku, tak by měly všechny indikátory zůstat vypnuté. Posledním kritériem, které zmíním, je výpočetní (tzn. časová a prostorová) náročnost algoritmu. Kvůli nutnosti identifikovat anomálie v reálném čase není možné, aby algoritmus trval déle než dvě průměrné delty. Volba dvou průměrných delt není arbitrární, zakládá se na faktu, že software přijímá packety v párech. Počet packetů, který při každém cyklickém spuštění softwaru přijmout je nastavitelný, ale packety zpravidla chodí v párech, a to v párech typu dotaz-odpověď. Volba dvou packetů se tedy zdá optimální. Z toho vyplývá, že průběh softwaru nesmí trvat déle než dvě průměrné delty, aby nedocházelo v lepším případě ke zpožďování detekce anomálií za generací packetů, a v horším případě k pádům softwaru.

Algoritmy byly dostatečně probrány a vysvětleny v kapitolách 2.5 a 3.2, proto v této části budou pouze zmíněny jejich výhody a nevýhody při implementaci v reálném čase. Dále bude vyčteno nastavení při implementaci. A finálně rozhodnuto, zda bude ve výsledném softwaru figurovat.

Statistické profilování

Jak již bylo několikrát zmíněno, tento algoritmus v principu funguje velmi jednoduše. Výpočet je rychlý a anomálie hledá efektivně. Nastavení, ke kterému jsem po opakovaných pokusech dospěl jako k optimálnímu je $k = 2{,}5$. Za nevýhodu tohoto se dá označit jeho inklinace k občasným falešně pozitivním výsledkům. Jmenovitě atribut Packet_Delta je v čase značně proměnlivý a jeho hodnoty se mohou vymykat nastaveným hranicím. Kvůli jednoduchosti a rychlosti výpočtu bude tento algoritmus figurovat ve výsledném softwaru.

Isolation Forest

Komplexnějším algoritmem, oproti algoritmu statistického profilování, časová, prostorová náročnost a kvalita výsledků Isolation Forest je silně závislá na jeho nastavení. Nejsložitější kalibrace ze všech laděných algoritmů patřila právě Isolation Forestu. S výsledným nastavením však spolehlivě odhaluje anomálie v našich packetech. Nastavení je následující: $\mathrm{n\_estimators} = 3;\ \mathrm{max\_samples} = 0{,}65;\ \mathrm{max\_features} = 1;\ \mathrm{n\_jobs} = -1;\ \mathrm{random\_state} = 1$ Posledním nastavením je již zmiňovaná contamination. Contamination se nastavuje v reálném čase v závislosti na výsledku statistického profilování. Pokud algoritmus statistického profilování neodhalí žádné anomálie, contamination je nastaveno na contamination = 0.01, neboli 1 % právě zkoumané sady dat. Toto nastavení je spíše citlivé než konzervativní a obdobně jako statistické profilování občas vykazuje falešně pozitivní indikace. Díky nastavení n_jobs = -1 může počítač vužívat všechna jádra procesoru ke stavbě stromů, klesá tak časová náročnost. Průběh je tedy rychlý, poměrově pouze dvakrát pomalejší než algoritmus statistického profilování. Díky dobrým výsledkům a rychlému průběhu si Isolation Forest zasloužil místo ve finálním algoritmu.

DBSCAN

Clustrovací algoritmus DBSCAN má, nepřekvapivě, závislou kvalitu výsledků na optimalizaci nastavení. Nastavovaných parametrů je v porovnání s Isolation Forest méně a tím je jeho ladění jednodušší. Po několika experimentech jsem jako optimální shledal nastavení $\mathrm{eps} = 0{,}8;\ \mathrm{min\_samples} = 3;\ \mathrm{algorithm} = \text{„}brute\text{“}$. V kapitole 3.2.3. byly vysvětleny parametry eps a min_samples dovysvětleme tedy parametr algorithm. Tento parametr určuje, jaký typ vzdálenosti bude DBSCAN uvažovat, když měří vzdálenost dvou bodů vůči sobě. Byly vyzkoušeny všechny možnosti a výsledky i časová náročnost byly nejlepší právě při tomto nastavení. I přesto

je DBSCAN časově náročným algoritmem, právě proto se nespouští v každé instanci Sliding Window, ale pouze když alespoň jeden z ostatních algoritmů identifikuje v sadě dat anomálii. Anomálie identifikuje naprosto spolehlivě a na falešně pozitivní indikace netrpí. Kvůli jeho přesnosti jsem ho shledal jako nutnost ve finálním softwaru.

ARIMA

Posledním zkoumaným algoritmem je ARIMA. Optimalizace parametrů v tomto případě je závislá na autokorelační a parciální autokorelační funkci. Je zřejmé, že Packet_Length, Packet_Delta a Packet_Value mají tyto funkce mezi sebou různé. Proto je nutné pro každý analyzovaný atribut packetů volat jinou verzi ARIMA. Po testování se však ukázalo, že časová náročnost algoritmu ARIMA je extrémně vysoká. Při běhu softwaru je nutné, aby celý výpočet trval kratší dobu, než dvě průměrné Packet_Delta. Toho nebylo možné v případě ARIMA dosáhnout. Zároveň se při vyšších řádech p,d,q parametrů časová náročnost ještě zvyšuje. Anomálie identifikuje správně nastavená ARIMA spolehlivě, trpí však ze stejných falešně pozitivních indikací jako statistické profilování a Isolation Forest. ARIMA není možné pro software použít z důvodu příliš vysoké časové náročnosti.

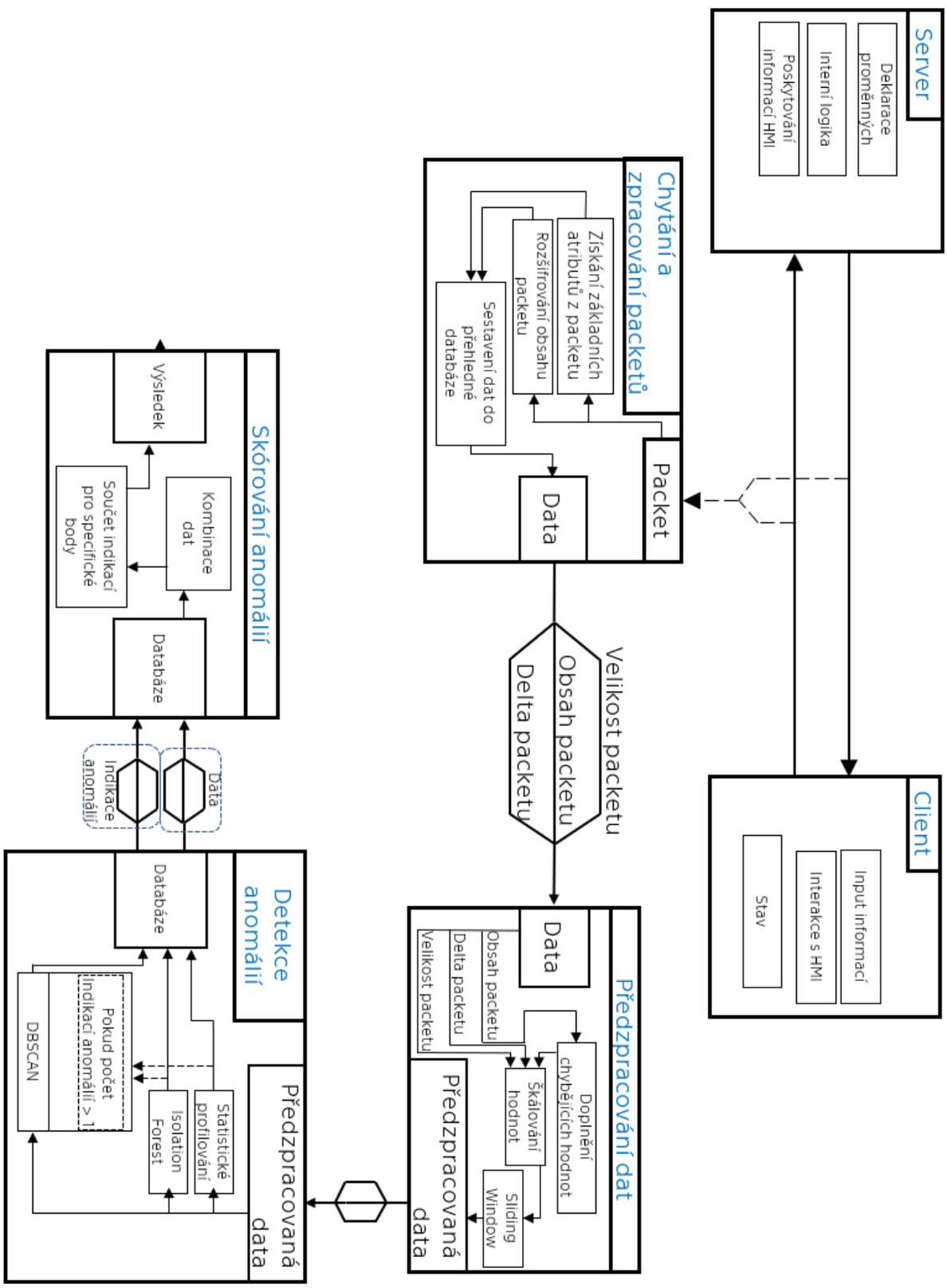
Výsledná kombinace

Vybrali jsme tedy 3 možné algoritmy, teď je na čase je spojit. Jak již bylo nastíněno, algoritmy běží sice paralelně, ale vzájemně se jejich průběh ovlivňuje. Parametr contamination u Isolation Forest je ovlivněn výsledkem statistického profilování a DBSCAN běží pouze pokud je anomálie identifikována jedním z předchozích algoritmů.

3.3.6. ARCHITEKTURA SOFTWARU

Algoritmy detekce anomálií nejsou zdaleka vším, o co se software musí starat. Další části softwaru jsou například již probraná dvojice serveru

s clientem, algoritmus na zachycování a dešifrování packetů, předzpracování dat a ve finále hodnocení anomálií. Ve zkratce si je představíme, přehledně je software rozebrán na Obrázek 23 - Diagram softwaru.



Obrázek 23 - Diagram softwaru

Chytání a zpracování packetu

Tato část softwaru odposlouchává komunikaci Server-Client pomocí knihovny Pyshark. Packety chytá po dvojcích a u každého zjišťuje atributy Packet_Length, Packet_Delta a Packet_Value. Tyto hodnoty ukládá do struktury DataFrame z knihovny Pandas a posílá dále na předzpracování.

Předzpracování dat

Předzpracování probíhá pro jednotlivé atributy odlišně. U atributu Packet_Value musí aktivně doplňovat chybějící hodnoty, protože ne každý packet s sebou nese informaci o nastavení pomyslné rychlosti odstředivky. Chybějící hodnoty jsou nahrazeny hodnotami předcházejícími. Dále ne všechny algoritmy detekce anomálií potřebují škálovat vstupní data. Z experimentů jsem usoudil, že jediný DBSCAN pro přesnost výsledků vyžaduje jako vstup normalizovaná data. Tudíž pro vstupy do Statistické profilování a Isolation Forestu používám nenormalizovaná data a pro vstup do DBSCANU data normalizovaná. Dále se tento podprogram stará o posílání vždy stejného objemu dat do samotných detekčních algoritmů. Objem posílaných packetů v každém časovém okamžiku je nastaven na posledních 16 přijatých packetů.

Skórování anomálií

Vstupem do tohoto podprogramu jsou všechna data z posledních 16 packetů a také indikace anomálií z jednotlivých algoritmů. Skórovací podprogram všechna tato data shromažďuje do jednoho DataFrame. Příklad výsledného DataFrame v Tabulka 3.

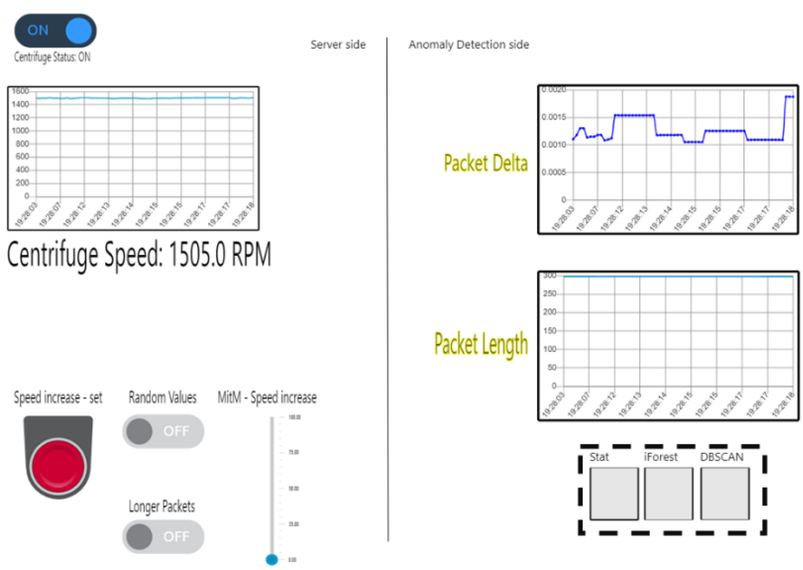
Index	Packet Length	Packet Delta	Packet Value	Anomaly Score	Length Anomaly	Delta Anomaly	Value Anomaly	Statistical Anomaly	IsolationForest Anomaly	DBSCAN Anomaly
3	202	0.08	1504	0	0	0	0	0	0	0
4	214	0.56	1503	3	0	1	0	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
19	202	0.07	700	2	0	0	1	1	0	1
20	214	0.1	1498	0	0	0	0	0	0	0

Tabulka 3

Výsledný DataFrame je v reálném čase ukládán do formátu „.csv“, ze kterého si server bere data potřebná k vizualizaci.

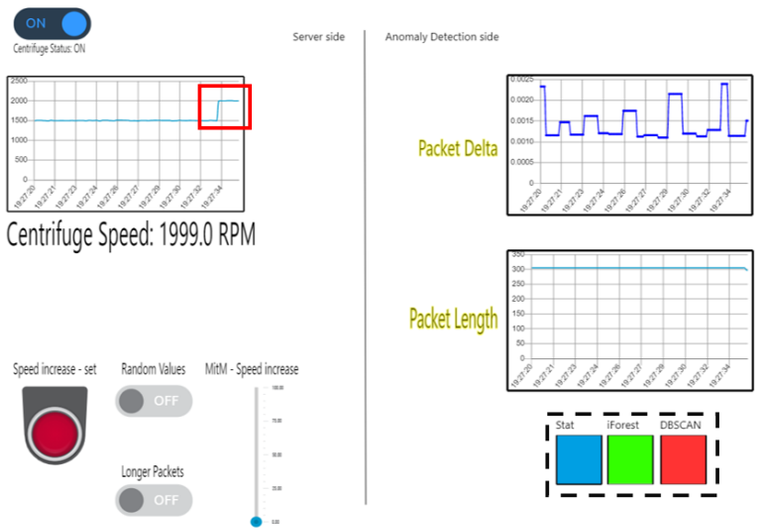
3.3.7. UKÁZKA VÝSLEDKŮ

Nejdříve si ukažme, jak vypadá běžný provoz bez anomálií. Na Obrázek 24 můžeme vidět, že při běžném provozu se nerozsvěcí ani jedna z kontrolek detekčních algoritmů.



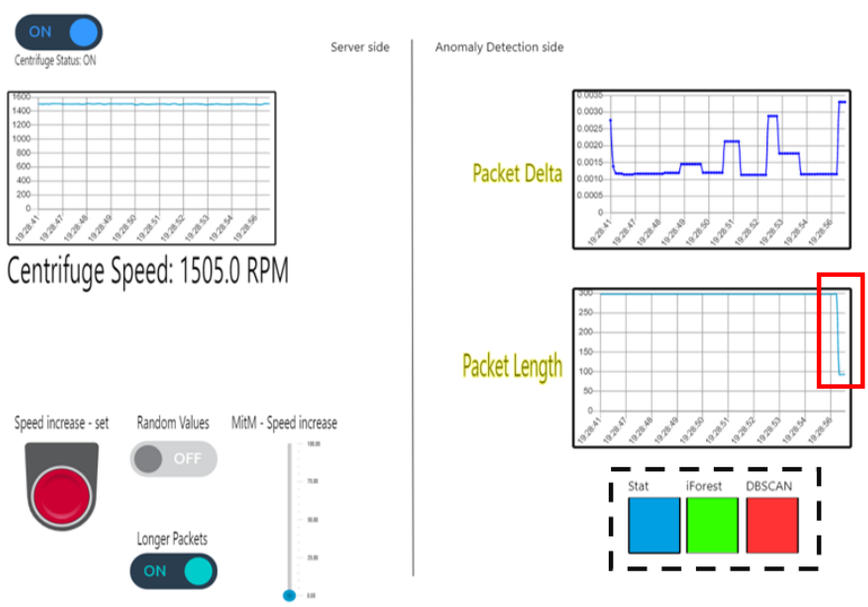
Obrázek 24

Dále spusťme různé typy anomálií a sledujme, zda to rozpoznají jednotlivé detekční algoritmy. Na Obrázek 25 - Zvýšení rychlost jsem skokově změnil hodnotu atributu Packet_Value o 25 %, všechny algoritmy tuto anomálii odhalily.



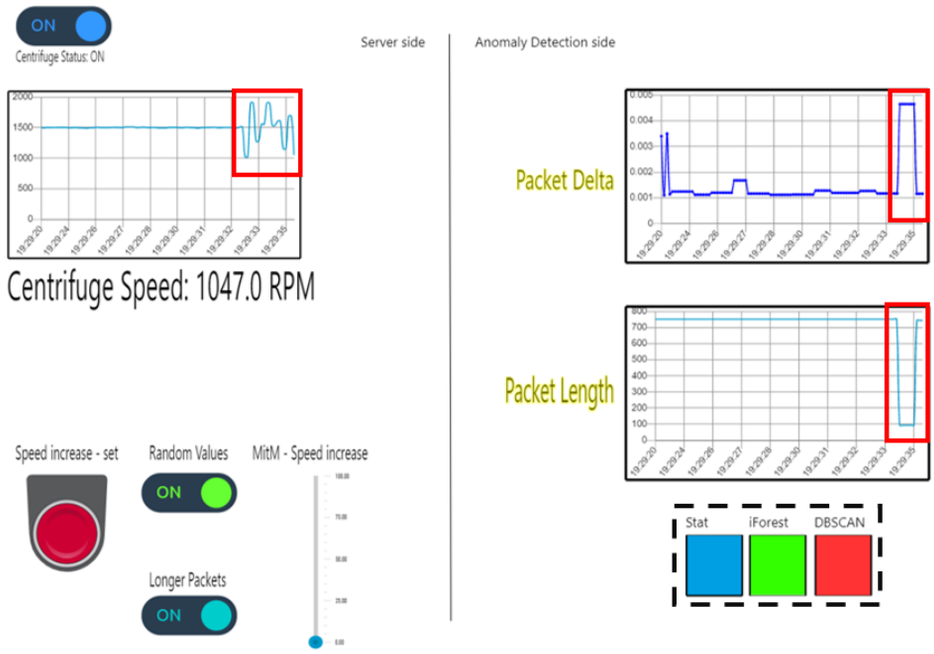
Obrázek 25 - Zvýšení rychlosti

Dále na Obrázek 26 - Kratší packetdošlo k náhlému snížení velikosti packetu, opět všechny algoritmy tuto anomálii odhalily.



Obrázek 26 - Kratší packet

Není tedy překvapivé že v situaci na Obrázek 27- více anomálií, když se objevily anomálie ve všech sledovaných atributech, tak všechny algoritmy opět tyto anomálie odhalily.



Obrázek 27- více anomálií

ZÁVĚR

Cílem diplomové práce bylo vytvořit prototyp softwarového díla pro detekci anomálií a možných útoků na komunikaci v průmyslových sítích. Tohoto bylo docíleno v programovacím jazyku Python3. Software je schopný úspěšně monitorovat komunikaci serveru s clientem v reálném čase, a hledat v ní anomálie. U metody statistického profilování se objevují instance, kdy falešně pozitivně indikuje anomálii. Stejně tak algoritmus Isolation Forest. U obou zmíněných se falešně pozitivní indikace objevují zejména při atypickém, ne však anomálním chování jednoho z monitorovaných atributů. DBSCAN na falešně pozitivní indikace netrpí. Při simulacích útoku software bezpečně indikuje všechny anomálie.

Moderní trendy ukazují na stupňující se počet útoků na industriální počítačové sítě, tudíž budou softwary, jako je tento prototyp, ve stále větší poptávce. Zaměstnanci pracující na kterékoliv laťce automatizační pyramidy, by měli do budoucna začít adoptovat bezpečnostní trendy popsané v této diplomové práci. Know-how a nový vhled, který jsem do problematiky získal při psaní této práce, bude bezpochyby v tomto směru užitečný.

Použité zdroje

[1] Behrendt, Andreas, N. Müller, P. Odenwälder a C. Schmitz, „McKinsey & Company," 30 March 2017. [Online]. Available: https://www.mckinsey.com/business-functions/operations/our-insights/industry-4-0-demystified-leans-next-level#.

[2] Cisco, „Cisco Annual Cybersecurity Report," Cisco, 2018.

[3] B. Galloway a G. Hancke, „Introduction to Industrial Control Networks," *IEEE Communications, Surveys & Tutorials,* sv. 15, č. 2, pp. 860-880, 2013.

[4] K. Stouffer, P. Victoria, S. Lightman, M. Abrams a A. Hahn, Guide to Industrial Control Systems Security, National Institute of Standards and Technology, U.S. Department of Commerce, 2015.

[5] „Javatpoint - DDoS," [Online]. Available: https://www.javatpoint.com/what-is-ddos-attack.

[6] B. Krebs, „Krebs on security," 30 November 2016. [Online]. Available: https://web.archive.org/web/20161220161008/https://krebsonsecurity.com/2016/11/new-mirai-worm-knocks-900k-germans-offline/.

[7] S. Gallagher, „Ars Technica," 21 October 2016. [Online]. Available: https://arstechnica.com/information-technology/2016/10/double-dip-internet-of-things-botnet-attack-felt-across-the-internet/.

[8] K. P. T. P. Eigner Oliver, „Detection of Man-in-the-Middle Attacks on Industrial Control Networks," v *International Conference on Software Security and Assurance*, St. Pölten, 2016.

[9] K. Zetter, „Wired," 11 November 2014. [Online]. Available: https://www.wired.com/2014/11/what-is-a-zero-day/.

[10] N. Falliere, L. O Murchu a E. Chien, „W32.Stuxnet Dossier," Symantec, 2010.

[11] A. Cui, „The Overlooked Problem of 'N-Day' Vulnerabilities," 26 March 2018. [Online]. Available: https://www.darkreading.com/vulnerabilities---threats/the-overlooked-problem-of-n-day-vulnerabilities/a/d-id/1331348.

[12] J. Šíma a R. Neruda, Theoretical Issues of Neural Networks, Prague: MATFYZPRESS, 1996.

[13] M. M. Gupta, I. Bukovsky, H. Noriyasu a e. al., „Fundamentals of Higher Order Neural Networks for Modeling and Simulation," v *Artificial Higher Order Neural Networks for Modeling and Simulation*, 2012, p. Chapter 6.

[14] I. Bukovsky, M. M. Gupta a S. Redlapalli, „Quadratic and cubic neural units for identification and fast state feedback control of unknown nonlinear dynamic systems," v *Fourth International Symposium on Uncertainty Modeling and Analysis*, 2003.

[15] A. Sharma V, „Understanding Activation Functions in Neural Networks," *medium.com,* 2017.

[16] S. Mainkar, „Gradient Descent in Python," 2018. [Online]. Available: https://towardsdatascience.com/gradient-descent-in-python-a0d07285742f.

[17] C. Wang, K. Viswanathan a et al., „12th IFIP/IEEE International Symposium on Integrated Network Management," v *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, Dublin, 2011.

[18] *I, Weston.pace, CC BY-SA 3.0 <http://creativecommons.org/licenses/by-sa/3.0/>, via Wikimedia Commons*, 2007.

[19] M. Masum, „Towards Data Science," 13 August 2020. [Online]. Available: https://towardsdatascience.com/identifying-ar-and-ma-terms-using-acf-and-pacf-plots-in-time-series-forecasting-ccb9fd073db8.

[20] A. Katchova, Econometrics Academy, 2013. [Online]. Available: https://sites.google.com/site/econometricsacademy/econometrics-models/time-series-arima-models.

[21] E. Sharova, „Unsupervised Anomaly Detection with Isolation Forest," v *PyData*, London, 2018.

[22] C. Mougan, „Isolation Forest from scratch," *Towards Data Science*, 22 May 2020.

[23] T. L. Fei, M. T. Kai a Z.-H. Zhou, „Isolation Forest," v *IEEE International Conference on Data Mining*, Pisa, 2008.

[24] A. Bhattacharya, „Effective Approaches for Time Series Anomaly Detection," 1 June 2020. [Online]. Available: https://towardsdatascience.com/effective-approaches-for-time-series-anomaly-detection-9485b40077f1.

[25] W. Badr, „Auto-Encoder: What is it? And What Is It Used For? (Part 1),“ *Towards Data Science*, 22 April 2019.

[26] L. Weng, „From Autoencoder to Beta-VAE,“ 12 August 2018. [Online]. Available: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html.

[27] C. Lupum, „www.kaggle.com,“ Intel Berkley Research Lab, November 2020. [Online]. Available: https://www.kaggle.com/caesarlupum/iot-sensordata.

[28] scikit-learn, „https://scikit-learn.org/,“ [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html. [Přístup získán 2021].