УНИВЕРЗИТЕТ
У НОВОМ САДУ
**ФАКУЛТЕТ ТЕХНИЧКИХ
НАУКА У НОВОМ САДУ**

Момчило Мицић

НАПАД НА SCADA СИСТЕМ

ПРОЈЕКАТ

Основне академске студије

Нови Сад, 2024.

Садржај

1. Увод

Циљ овог пројекта је био да се стекне увид у различите начине на које се осмишљава један SCADA (*Supervisory Control And Data Acquisition*) систем, те различити приступи нападања и одбране оваквих система.

SCADA систем је систем међусобно повезаних електронских компоненти које контролишу неки процес, нпр. производну траку, и омогућава праћење и контролу над тим процесом [1, 7]. Такав систем се обично састоји од конкретног процеса који има своје улазе и излазе, који се прате преко сензора и актуатора. Информације које они сакупљају интерпретирају PLC-ови (*Programmable Language Controllers*). Уз помоћ мрежних протокола те информације се шаљу удаљеном рачунару (SCADA станцица) како би одговорно лице или више њих имали увид у стање процеса. Такви системи су у основи надзора система критичних инфраструктура, као што су електродистрибуција, водовод итд. Као такви, мета су нападача који на разне начине покушавају да добију неку корист од тога.

Зато смо се ми фокусирали на то да добијемо искуство у развоју таквих апликација, нападању и њиховој одбрани. Конкретно, при изради пројекта су постојале 3 фазе:

1. Имплементација MODBUS протокола и осмишљавање конкретног SCADA система,
2. Вршење напада
3. Коришћење вештачке интелигенције приликом детекције напада

Наш конкретни SCADA систем представљао је део нуклеарног реактора са водом под притиском, где су симулиране контролне шипке и мерач температуре воде. Напади су били *Command Injection* и *Replay Attack*, по узору на рачунарски вирус *Stuxnet* који је у периоду од чак 2005. до 2010. године лежао неоткривен на рачунарима иранских нуклеарних постројења [2]. Алгоритам за детекцију напада је био *Random Forest* који се ослања на *Decision Tree*.

2. Опис коришћених технологија и алата

У пројекту су коришћене две главне технологије, *.NET Framework* 4 и *Python* 3 са проширењима у зависности од потребе.

2.1. .NET Framework 4

.NET Framework 4 је скуп алата за израду *Windows* апликација у разним језицима, међу којима су *VisualBasic.NET*, C#, F#, J# итд. Сви се они преводе на *Common Language* бајт код чије извршавање у CLR (*Common Language Runtime*) омогућава аутоматско сакупљање непочишћене меморије и извршавање апликације [3]. Самим тим C# је језик релативно високих перформанси који омогућава израду комплексних desktop апликација, уз могућност лаког дебаговања и коришћења функција нижег нивоа. Као такав је изабран за развој симулатора постројења због претходног искуства у раду са *.NET Framework*-ом. Такође подржава и *Berkeley Sockets* API (*Application Programming Interface*) који се користи за мрежну комуникацију [4]. Тиме је елиминисана потреба за посебним начином комуникације и олакшана 1 на 1 имплементација MODBUS протокола, јер и *Python* подржава *Socket*-е.

2.2. Python 3

Python 3 је интерпретирани програмски језик који омогућава брзо, једноставно, флексибилно и лако прављење простих, па и комплекснијих апликација, наштрб перформанси [5]. *PyQt5* је пакет проширења који уводи *Python* wrapper за међуплатформску C++ библиотеку *Qt5* која омогућава лакшу израду графичког корисничког интерфејса [6]. *PyDivert* је проширење које уводи wrapper за *WinDivert* драјвер који омогућава апликацијама да прикупљају и измењују пакете које пролазе кроз мрежну картицу рачунара [10, 11]. *Pandas*, *Scikit-Learn*, *NumPy* и *XGBoost* су *Python* библиотеке које омогућавају лакшу обраду велике количине података који се могу користити за тренирање алгоритама машинског учења [14].

Апликација SCADA надзорне станице је израђења у *Python*-у са *PyQt5* проширењем а са симулатором постројења комуницира преко *Socket* API.

Апликација која врши прислушкивање на мрежи и измену пакета имплементирана је помоћу *PyDivert* библиотеке.

Модул који је искоришћен за детекцију напада у SCADA надзорној станици јесте модел трениран уз помоћ *XGBoost* заједно са *Scikit-learn*, а подаци за тренирање прикупљени су и обрађени уз помоћ *Pandas* и *NumPy* библиотека.

3. Опис решења проблема

Како је свака апликација рађена од нуле, изградњи пројекта се морало приступити слојевито, у више фаза, где је свака наредна фаза користила сазнања из претходне. Прво се приступило основама, што је у нашем случају била 1:1 имплементација SCADA MODBUS протокола. MODBUS протокол је морао бити имплементиран и на страни надзорно-управљачке станице и на страни симулатора постројења. Након успешне имплементације и тестирања стабилности SCADA станице и симулатора, приступило се осмишљавању нашег постројења, те нападању и напослетку детекцији напада.

3.1. MODUBS протокол

MODBUS протокол је један од стандардних протокола комуницирања SCADA надзорне станице и PLC (*Programmable Logic Controller*) уређаја који сакупљају свеже податке са сензора и шаљу их ка надзорној станици [7 – 9].

Имплементира се на апликативном нивоу мрежног комуникационог stack-а, изнад TCP (*Transaction Control Protocol*) и заснован је на клијент-сервер тј. *master-slave* моделу комуникације. Надзорна станица представља клијент илити *master* и она у одређеном интервалу задаје упите за читање или упис нових вредности ка процесним контролерима, које они даље прослеђују на сензоре или актуаторе респективно. Сервери тј. у овом случају процесни контролери одговарају томе које су вредности прочитане, или у случају уписа, које су вредности уписане [7 – 9].

Меморијске локације у којим се налазе вредности које описују стање постројења зову се регистри. У зависности од тога да ли они представљају вредности сензора или актуатора (да ли само прикупљамо податке или утичемо на промену стања), они могу бити улазни (само читање) или излазни регистри (дозвољено и читање и писање). Они се даље деле на: аналогни излаз, аналогни улаз, дигитални излаз, дигитални улаз. Дигитални регистри представљају вредности прекидача, тј. могу имати само вредности 0 или 1, док аналогни могу имати више вредности. Сваки од тих регистара се налазе на посебним меморијским адресама [7].

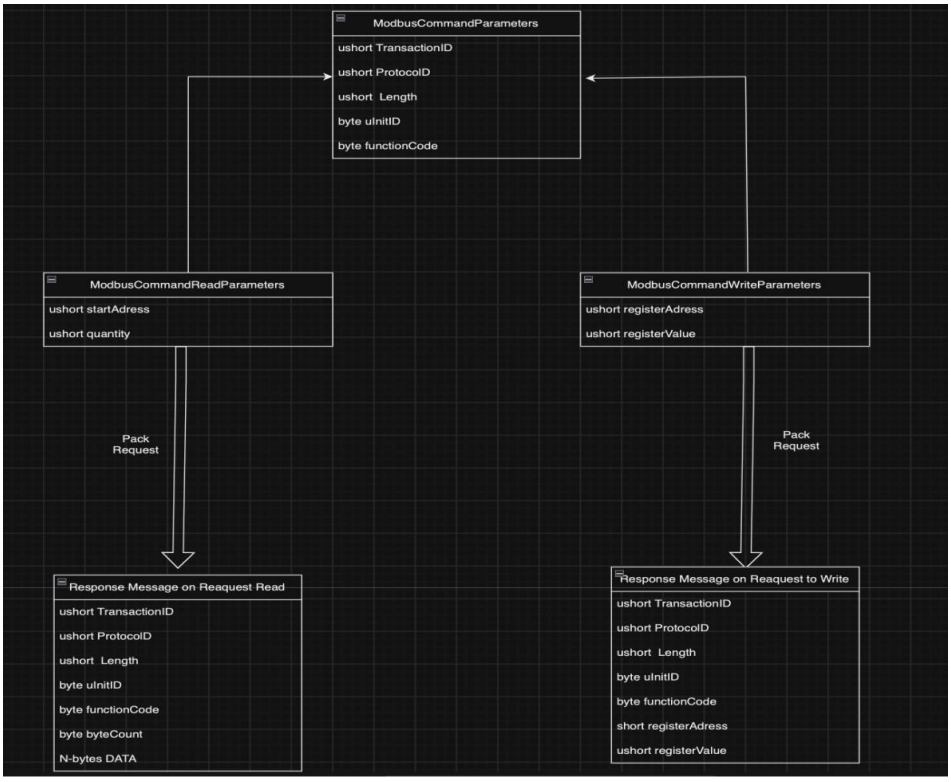
Надзорна станица и контролери комуницирају разменом мрежних пакета, при чему се разликују у складу са врстом регистра којима су упућени, и од тога да ли је пакет везан за упит или одговор. Сваки MODBUS *Request* пакет садржи:

1. *TransactionID* – поље величине 2 бајта, и говори који је редни број пакета тј. трансакције
2. *ProtocolID* – говори који протокол користимо, у нашем случају ће ова вредност увек бити 0, а због различитих верзија MODBUS протокола овај број може да се разликује. Поље је величине 2 бајта
3. *Length* – представља број бајтова који заузимају наредна поља
4. *UnitID* – представља број станице са које вршимо прикупљање података. У нашем случају одабрали смо број 100. Овај број нам је доста значајан уколико прикупљамо и приказујемо податке на више различитих SCADA станица. Величине је једног бајта
5. *FunctionCode* – говори нам коју операцију желимо да извршимо на процесном контролеру и величине је једног бајта

Вредности *FunctionCode*-а осликавају прави стандард MODBUS протокола и оне могу бити:

1. Прочитај дигитални излаз - 0x01
2. Прочитај дигитални улаз - 0x02
3. Прочитај аналогни излаз - 0x03
4. Прочитај аналогни улаз - 0x04
5. Упиши дигиталну вредност - 0x05
6. Упиши аналогну вредност - 0x06

Овакав пакет смо пресликали у класи *ModbusBase* у нашем софтверу. У зависности од кода функције, у имплементацији имамо различите класе које осликавају све врсте пакета. На слици 3.1 може се видети дијаграм наслеђивања од примљеног пакета и како изгледа одговор.



Слика 3.1. Дијаграм наслеђивања класа MODBUS пакета

Ако је у питању *ReadRequest*, онда су остала поља:

1. *StartAddress* – величине 2 бајта и њиме се дефинише почетна адреса регистара које желимо да прочитамо
2. *Quantity* – исто величине 2 бајта и представља број узастопних локација које желимо да прочитамо, пошто за аналогне регистре треба више бајтова

На *ReadRequest* добијамо *ReadResponse* чија су поља *TransactionID*, *ProtocolID*, *UnitID* и *FunctionCode* потпуно иста и имају исте вредности док се Length мења у зависности од потребе, а додатна поља су:

1. *ByteCount* – представља број прочитаних бајтова, поље величине једног бајта
2. *Data* – представља низ бајтова прочитане вредности у складу са захтевом

Ако пак желимо да упишемо нешто, тј. да променимо стање система, послаћемо *WriteRequest* који садржи иста поља као *ModbusBase* са додатком:

1. *RegisterAddress* – величине 2 бајта и представља адресу на коју желимо да упишемо вредност
2. *RegisterValue* – величине 2 бајта, представља тачну вредност коју желимо да упишемо

На *ReadRequest* добијамо од симулатора / PLC-а *ReadResponse* који је у потпуности исти као захтев за упис јер треба да осликава да су вредности лепо уписане. Уколико се то не деси, постоји и систем слања изузетака где ће уместо разних горенаведених поља бити послати кодови за изузетке, који за потребе пројекта нису толико битни али су свакако имплементирани у пројекту и са стране надзорне станице и са стране симулатора постројења.

3.2. Симулатор постројења

Симулатор постројења израђен је као C# конзолна апликација која обрађује горенаведене пакете од SCADA станице, и шаље јој одговоре, притом симулирајући процесно постројење које се могло програмски осмислити. Самим тим је симулатор било логично саградити из два главна дела: мрежни део и део постројења.

Мрежни део прима, обрађује и шаље пакете и у односу на њих мења унутрашње стање система. Део постројења је имплементиран као *ConcurrentDictionary* где је кључ адреса регистра који представља вредност мерног уређаја или актуатора, а вредност сам тај уређај. Стање постројења се мења у бесконачној петљи која одговара томе како се требају мењати стања у неком реалном систему, и ову петљу је могуће програмирати да реплицира понашање неког реалног постројења. За потребе лакшег дебаговања и увиђаја у ефикасност напада конзална апликација стално исписује вредности на свим регистрима.

3.3. SCADA HMI (*Human-Machine Interface*)

SCADA HMI или SCADA станица је *Python* апликација са GUI (*Graphical User Interface*) компонентом имплементираном помоћу *PyQt5* која у табеларном

облику приказује вредности свих регистара које прочита са симулатора постројења, по узору на реалне SCADA станице како бисмо имали тачан увид у стање система. На слици 3.2 приказан је изглед апликације. Вредности које постоје у систему се могу унети у конфигурациони фајл који ће се очитати при покретању апликације.

Слика 3.2. Покреунта надзорно-управљачка станица, са табеларним приказом стања система (горе), индикатором повезаности са симулатором (доле лево) и имдикатором абнормалног стања (доле десно)

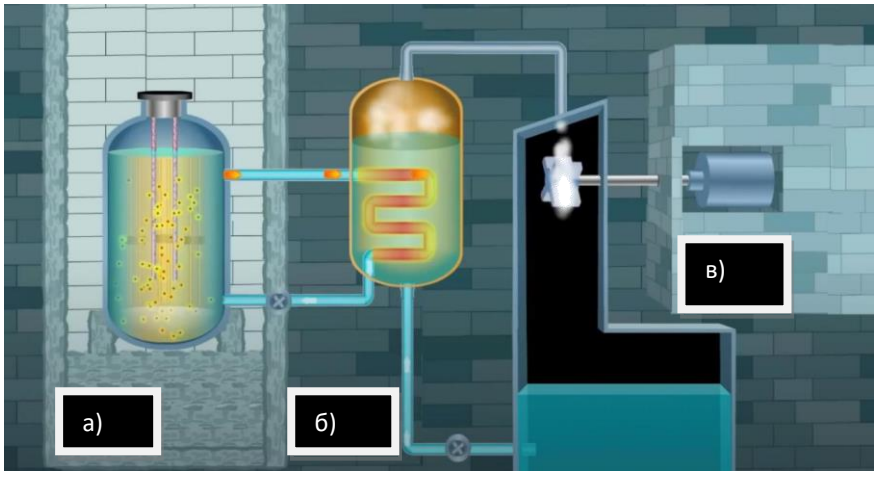
Апликација на одређеном временском интервалу обавља циклус аквизиције података, након чега их обради, и у зависности од програмираног начина управљања, послаће команде за измену стања симулатору постројења. Осмишљавању самог постројења ћемо приступити након правилне имплементације ове две апликације.

Како су симулатор и надзорно-управљачка станица самосталне у реалном свету, тако су и код нас одвојене апликације и једна може да ради без друге, док се не покрену обе истовремено неће бити успостављена веза између њих. Зато су имплементирани безбедносни механизми да не би долазило до пуцања апликације уколико једна од страна не буде имала конекцију ка другој.

3.4. Осмишљавање постројења

Било нам је потребно неко постројење које је занимљиво људима који ће гледати пројекат, а да је у исто време лако за разумевање и имплементацију. Стога смо се одлучили да моделујемо део нуклеарног реактора са водом под притиском (PWR – *Pressurized Water Reactor*).

Нуклеарни реактори служе за производњу велике количине електричне енергије, тако што помоћу нуклеарне енергије греју воду која се претвара у пару и под притиском гура турбине генератора који генерише струју, као на слици 3.3. Нуклеарна енергија у реакторима са водом под притиском добија се из тзв. решетки горива (*Fuel Rods*) које су направљене од веома радиоактивних елемената, обично неког изотопа уранијума. Решетке уранијума су потопљене у воду која служи као посредник којим се греје водена пара која покреће генератор. Саме решетке уранијума су толико радиоактивне јер избацују огромну количину неутрона који изазивају ланчану реакцију цепања других језгара и тиме се ствара огромна количина енергије која мора бити изконтролисана. Уколико се та енергија некако не изконтролише, може доћи до огромне температуре у резервоару са водом а последице тога могу бити катастрофалне (топљење резервоара, радијација излази напоље итд.). Зато постоје тзв. контролне решетке које упијају неку количину неутрона које избаце решетке уранијума и тиме смањују температуру воде у резервоару. Контролне решетке су обично изграђене од бора или кадмијума [15 – 16].



Слика 3.3 Упрошћени приказ нуклеарног реактора са водом под притиском. Резервоар са контролним и решеткама за гориво је под а). Вода која се претвара у пару је под б), док је в) генератор.

Услед недостатка времена и већ довољне комплексности пројекта, одлучили смо да упростимо ово постројење тако што ћемо да га представимо помоћу два уређаја, једног сензора и једног актуатора. Сензор ће представљати стање термометра који мери температуру воде унутар резервоара. Актуатор ће представљати позицију контролних решетки у односу на воду. Када је на вредности 1, контролне решетке су спуштене у воду и успоравају реакцију и хладе воду, а када су на вредности 0, оне су изван воде и тиме се реакција повећава а самим тим и температура. Актуатор би периодично требало палити и гасити у зависности од температуре воде која треба да се одржава у границама од 250 до 350 степени целзијусове скале.

Овим смо искористили и аналогне и дигиталне регистре, и оне из којих се може читати и уписивати, те и да смо узели некакав комплекснији систем, апликације би свакако радиле на исти начин, само би се процес будућег тренирања неуронске мреже беспотребно закомпликовати.

3.5. Нападање система

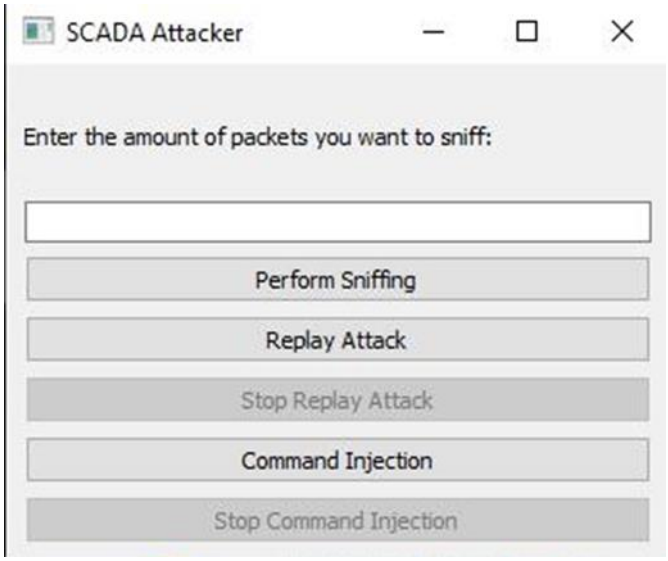
Пошто, као и у реалном свету, надзорно-управљачка станица и процесни контролери комуницирају путем TCP-а [9], логичан след догађаја из угла нападача био би да прислушкујемо TCP пакете и покушамо да откријемо оне који подесћају на MODBUS пакете. Када утврдимо на којим портовима се одвија комуникација, следи прислушкивање тих портова.

Главна идеја око нападања јесте да прво прислушкивањем сакупимо неколико пакета са подацима са сензора и актуатора. Главни напади које бисмо са добијеним информацијама могли да извршавамо су *Replay Attack* и *Command Injection*.

Replay Attack је врста напада где неком систему шаљемо застареле податке како бисмо га збунили [17]. Наша имплементација овог напада огледа се у томе да податке са сензора које смо прикупили прислушкивањем шаљемо уместо правих података који стижу са симулатора постројења. То постижемо тако што ћемо ухватити сваки пакет чији је одредишни порт порт од надзорно-управљачке станице и у њему заменити тренутну вредност са вредношћу сакупљеном током прислушкивања. Тиме ћемо надзорној станици слати лажне податке, и она ће подешавати актуатор у односу на њих, што може направити велику невољу у стварном систему.

Command Injection или инјекција команди се односи на врсту напада код које се неком систему шаљу команде које не би смеле да се шаљу, или се прослеђују погрешни параметри [18]. Код нас, то значи да бисмо узимали пакете који упућују команду спуштања или подизања контролних шипки и инвертовали јој вредност и тиме директно утицали на процесне контролере да намерно мењају температуру реактора и потенцијално прегрејали га. Међутим, ако бисмо само мењали пакете захтева уписа, одговори би враћали стварно стање система и то би се голим оком видело на табели да нешто није у реду. Тако да смо ми морали мењати и све пакете читања са тог актуатора да би напад био неприметан.

Тако смо нашу нападачку апликацију развили уз помоћ *PyDivert* пакета у *Python*-у. Пошто *PyDivert* заправо обухвата *WinDivert* драјвер, то значи да би нападачка апликација могла савршено да ради и у облику неког DLL (*Dynamic Link Library*) фајла ради тежег откривања, али смо ми искористили *Python* ради лакшег и бржег развоја и тестирања. Наша апликација прво прикупља последњих неколико десетина пакета послатих од стране симулатора постројења и чува их у фајлу. Потом можемо покренути *Replay Attack* или *Command Injection* напад. На слици 3.4 је приказан кориснички интерфејс апликације.



Слика 3.4 Изглед апликације за нападање

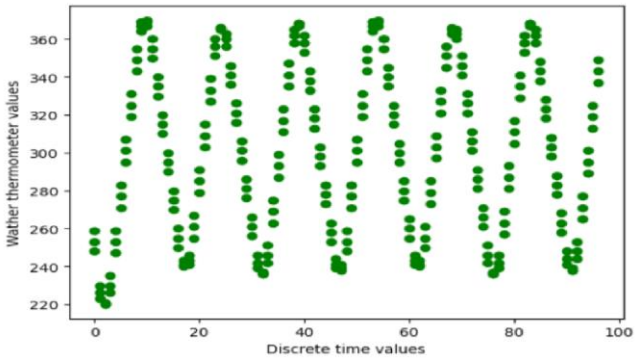
Replay Attack и *Command Injection* напади извршени самостално се релативно лако откривају, тако да наша апликација нуди могућност извршавања оба напада истовремено како бисмо потпуно уништили функционалност система.

3.6. Детекција напада

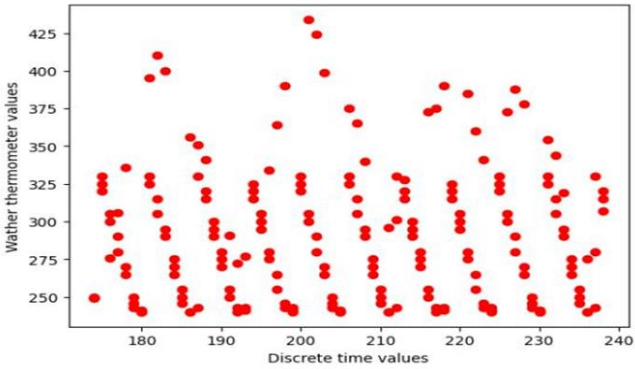
Размотрили смо више начина којима бисмо помоћу машинског учења детектовали аномалије у вредностима на надзорној станици. Неки од начина су биле неуронске мреже и алгоритме базиране на одлукама.

Неуронске мреже су широко распрострањене у различитим сферама науке, поготово науке која се бави подацима (*data science*) и имају широку примену зато што њихово функционисање наликује на људске неуроне. Међутим, оне су теже за тренирање јер су подесне за разне употребе. Наш проблем се свео на проблем класификације, где бисмо имали неколико стања система:

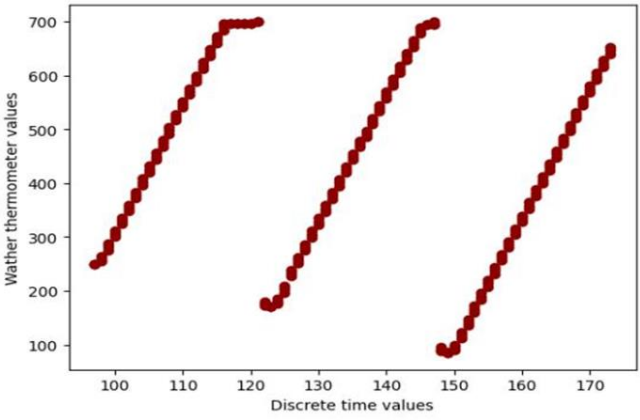
1. Нормално стање, приказано дијаграмом на слици 3.5
2. Стање *Replay Attack*, слика 3.6
3. Стање *Command Injection*, слика 3.7



Слика 3.5 Систем у нормалном режиму рада



Слика 3.6 Стање система када је активан Replay Attack



Слика 3.7 Стање система током Command Injection-а

Алгоритми који су мало бољи у класификовању већ познатих или структуираних случајева су тзв. алгоритми базирани на одлукама илити *decision support systems*. Међу такве алгоритме спада и *Random Forest*, који се заснива на стаблима одлуке, такозваним *Decision Tree*-овима [12].

У једном стаблу одлучивања, чворови представљају критеријуме по којима се процењује вероватноћа да неки податак спадне у неку класу. Шума се тренира тако што подешава тежине шанси да се неки податак класификује на неки начин, тј. више стабала се одједном тренирају, са тенденцијом да се прво изаберу стабла која су раније погрешно класификовала неки податак [12, 13]. Наша шума је тренирана у односу на последње три вредности које су примљене са сензора температуре.

На сликама 3.5, 3.6, и 3.7 се јасно виде разлике у примљеним вредностима приликом *Replay Attack*-а и *Command Injection* напада. Над тим подацима је тренирана наша насумична шума и помоћу *XGBoost*-а убачена као готов модел у апликацију SCADA станице, тако да у реалном времену упозорава корисника на неадекватне вредности у систему.

4. Закључак

Израдом овог пројекта добили смо увид у начине размишљања и приступ проблему нападања и заштите система критичне инфраструктуре, успут истражујући могућности које нам нуде технике машинског учења у превенцији штете над неким системом. Стекли смо искуство у имплементирању апликација на различитим *framework*-овима и успешно их повезали.

4.1. Идеје за даља унапређења

Поред генералног побољшања стабилности и рефакторисања кода апликације, главна унапређења би се могла свести на:

1. Усложњавање постројења и детаљнија симулација – Тренутни модел нукеарног постројења је, иако ефикасан за доказивање поенте пројекта, релативно прост. Постоји могућност да се направи детаљнији модел заснован на правим законима физике и математике који би можда могао и графички да се симулира у симулатору постројења. То би допринело интерактивности пројекта.
2. График историјата – Додавањем графика на ком се јасно виде претходне измерене вредности на SCADA станици, апликација би била комплетнија и личила на праву SCADA апликацију.
3. Енкрипција пакета – Коришћењем асиметричног алгоритма енкрипције порука, као на пример RSA (*Rivest–Shamir–Adleman*), између надзорно-управљачке станице и симулатора постројења у великој мери отежала би се могућност прислушкивања и измене пакета.
4. Промена параметра машинског учења – Ако бисмо уместо тачних вредности сензора или актуатора користили извод или обе мере при тренирању, добили бисмо веродостојнију и бржу реакцију алгоритма за детекцију на аномалије у прочитаним вредностима.

Литература

[1] SCADA International. SCADA systems explained. [Online] Доступно на: https://scada-international.com/what-is-scada/

[2] Symantec. W32.Stuxnet Dossier (November 2010). [Online] Доступно на: https://www.wired.com/images_blogs/threatlevel/2010/11/w32_stuxnet_dossier.pdf

[3] Microsoft. Overview of .NET Framework (2023). [Online] Доступно на: https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview

[4] IEEE Std. 1003.1-2001. IEEE Standards Association, 2001. [Online] Доступно на: https://standards.ieee.org/ieee/1003.1/1389/

[5] Python Software Foundation. Python standard library (2023) [Online] Доступно на: https://docs.python.org/3/library/index.html

[6] Riverbank Computing Ltd. PyQt5 Project description (Oct 14, 2023) [Online] Доступно на: https://pypi.org/project/PyQt5/

[7] Бранислав Атлагић. Софтвер са критичним одзивом – пројектовање SCADA система. Нови Сад, Србија: ФТН, 2015.

[8] Modbus. Modbus Application Protocol Specification (2012). [Online] Доступно на: https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[9] Modbus Organization. MODBUS Messaging on TCP/IP Implementation Guide V1.0b (2006). [Online] Доступно на: https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[10] Reqrypt. WinDivert (2023). [Online] Доступно на: https://reqrypt.org/windivert.html

[11] PyDivert Api Reference [Online] Доступно на: https://pythonhosted.org/pydivert/

[12] Stanford University Online. (May 14, 2021). CS229. [Online] Доступно на: https://cs229.stanford.edu/notes2021spring/notes2021spring/Decision_Trees_CS229.pdf

[13] Stanford University Online. (2021). DeepLearning.AI. [Online] Доступно на: https://github.com/dusvn/Machine-Learning-Specialization/blob/main/Advanced%20Learning%20Algorithms/C2_W4.pdf

[14] Xgboost developers. XGBoost Documentation (2022). [Online] Доступно на: https://xgboost.readthedocs.io/en/stable/

[15] Elearnin. Nuclear Reactor – Understanding how it works | Physics Elearnin. (Apr 23, 2013). [Online video]. Доступно на: https://www.youtube.com/watch?v=1U6Nzcv9Vws

[16] AREVA – Image and Process. Pressurized Water Reactor. (Geb 11, 2009). [Online video]. Доступно на: https://www.youtube.com/watch?v=MSFgmLW1Crw

[17] AO Kaspersky Lab. “What Is a Replay Attack?”. kaspersky.com. [Online.] Доступно на: https://www.kaspersky.com/resource-center/definitions/replay-attack
[18] Weilin Zhong. “Command Injection”. owasp.org. [Online.] Доступно на: https://owasp.org/www-community/attacks/Command_Injection

Списак коришћених слика и табела