



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ



Душан Париповић ПР76-2020  
Момчило Мицић ПР69-2020

**Напад на SCADA систем**  
**ПРОЈЕКАТ**  
- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 25.12.2023.

## САДРЖАЈ

1. ОПИС РЕШАВАНОГ ПРОБЛЕМА
2. ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА
3. ОПИС РЕШЕЊА ПРОБЛЕМА
4. ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА
5. ЛИТЕРАТУРА

## ОПИС РЕШАВАНОГ ПРОБЛЕМА

У решавању проблема имплементације Modbus протокола за размену порука између SCADA HMI (Supervisory Control and Data Acquisition - Human Machine Interface) система и постројења, фокус је стављен на пројектовање система са најмање једним сензором и једним актуатором. Симулирало се деловање нуклеарне електране користећи сензор за температуру воде и контролне решетке за управљање нуклеарном реакцијом и температуром воде. Први корак било је одређивање формата порука за аквизицију и управљање системом као и имплементација Modbus протокола [1][3]. Кључно је било утврдити са којих уређаја је могуће само читати тренутне вредности, а на које се може и уписивати или мењати стање из SCADA HMI станице [1][3]. Након постављања основа за комуникацију, фокус се пребацио на дефинисање аутоматизације система и симулације постројења. Са стране симулатора постројења, циљ је био осмислити начин да се симулирају физички закони унутар нуклеарне реакције и да се размотри веза између стања контролних решетки и температуре воде унутар реактора. Са стране SCADA HMI станице циљ је био омогућити систему аутоматско реаговање на одређене вредности, чиме се смањује потреба за константним ручним слањем команди од стране оператера. Систем је програмиран да шаље одговарајуће команде у зависности од детектованих вредности, што повећава ефикасност управљања [2]. У развоју система, кључно је било дефинисати low и high аларме, односно опсег температуре воде релевантан за процес аутоматизације. Разматрање граница температуре воде омогућило је прецизно програмирање система да реагује на ниске и високе вредности, доприносећи тако безбедности и оптималном раду постројења [2][3].

У фази имплементације, размишљало се као потенцијални нападачи, проучавајући детаљно Modbus протокол и идентификујући могуће рањивости у систему. Имплементирани су replay attack и command injection као два кључна типа напада. Replay attack представља технику где нападач снима и касније репродукује претходно ухваћени саобраћај да би обмануо систем. Овај тип напада је посебно опасан у контексту SCADA система, где неовлашћено понављање контролних команди може изазвати озбиљне последице. Command injection је врста напада где се злонамерне команде убацују у систем путем инпут поља или других механизма. У SCADA систему, command injection може бити посебно опасан јер нападач може покушати уметање злонамерних Modbus команди, што може резултовати извршавањем неовлашћених акција, променом параметара система, онеспособљавањем одређених функционалности или довођењем система у непожељно или неконзистентно стање.

У последњој фази пројекта примениле су се методе машинског учења да би се детектовали потенцијални напади на систем. Модели који су развијени анализирали су обрасце понашања у комуникацији између SCADA HMI система и постројења, идентификујући необичности које би могле указивати на злонамерне приступе.

У решавању изазова повезаних за replay attack и command injection нападе у оквиру Modbus протокола, примењене су различите стратегије и технике како би се ове претње свеле на минимум. Када је у питању одбрана од replay attack напада, често се користи приступ коришћења временских ознака (timestamps) при комуникацији између SCADA HMI и постројења. Свака команда или порука обележена је временском ознаком, а систем на постројењу проверава да ли су временски параметри у складу са очекиваним временским интервалима при примању порука. Уколико систем уочи необичне временске параметре, може препоставити replay attack и применити одговарајуће мере заштите, чиме се минимизује ризик од неовлашћених репликација. Што се тиче одбране од command injection напада, ефикасан приступ укључује размену енкриптованих порука уз помоћ алгоритама асиметричне криптографије, нпр. RSA алгоритам. Свака порука се шифрује уз помоћ јавног кључа и дешифрује уз помоћ приватног кључа. Такво шифравање порука ће последично омогућити и заштиту од разних других типова напада који укључују прислушкивање порука и њихову

измену. У раду се дискутује о методама и техникама које тимови користе у заштити система од replay attack и command injection напада, и истиче се значај правилне валидације, филтрације и временских ознака као ефикасних стратегија за минимизирање ризика од ових видова напада.

## ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА

Пројекат је имплементиран коришћењем различитих технологија, свака прилагођена специфичним захтевима и улози у систему.

### 1. Симулатор система у C# .NET-у:

- Развијен у програмском језику C# .NET.
- Конзолна апликација која симулира рад дела нуклеарне електране.

### 2. SCADA HMI у Python-у са PyQt5:

- Python коришћен за имплементацију SCADA HMI.
- PyQt5 библиотека употребљена за креирање корисничког интерфејса.
- Python одабран због флексибилности и једноставности у развоју интерфејса.

### 3. Man in the Middle компонента са PyDivert библиотеком [5]:

- Имплементирана као third-party апликација.
- PyDivert библиотека користи се за пресретање пакета и извођење напада на систем [5].
- Пружа функционалност праћења и манипулације саобраћајем између SCADA HMI и симулатора система.

### 4. Комуникација између апликација путем SOCKET-а и TCP протокола:

- За комуникацију између делова система коришћени су Berkeley Sockets API и TCP протокол.
- Са овим видом комуникације показали смо да нам је сасвим небитно у ком програмском језику су написане наше 2 апликације, једино што је битно је како ми уписујемо бајтове и како их читамо.
- Поштовањем одређених процедура постигли смо успешну комуникацију.

### 5. Машинско учење у Python-у са Pandas, Scikit-learn, NumPy и XGBoost [6][7]:

- Библиотеке као што су pandas, scikit-learn, numpy и xgboost коришћене су у оквиру SCADA HMI [6][7].
- Имплементиране за анализу података и развој модела машинског учења.
- Додају слој сигурности и детекције претњи кроз машинско учење.

## ОПИС РЕШЕЊА ПРОБЛЕМА

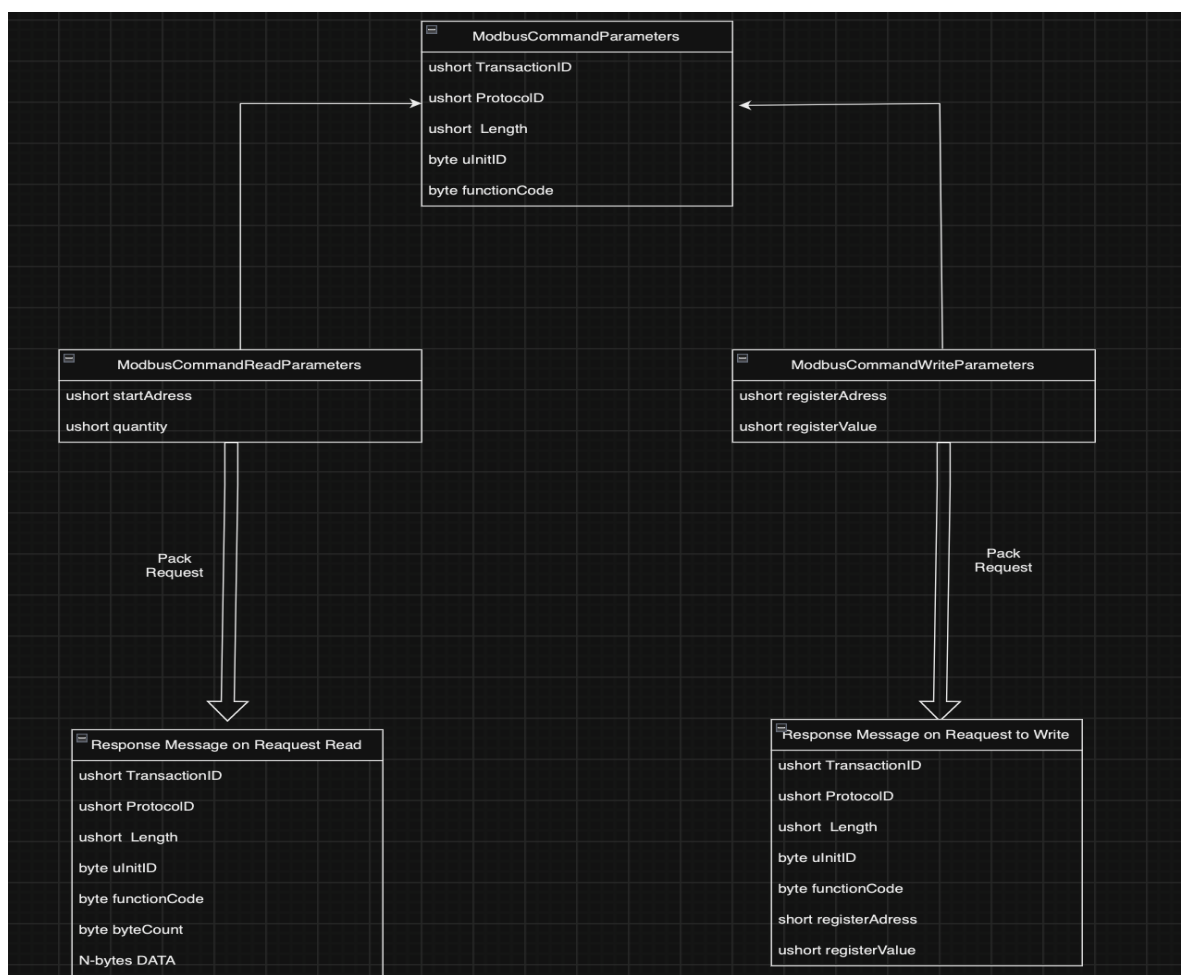
Само решавање проблема се састојало из више фаза. У даљем наставку текста проћи ћемо поступно кроз сваку фазу и потрудити се што боље то да приближимо.

### Фазе развоја:

1. Имплементација Modbus протокола и дефинисање система
2. Развој корисничног интерфејса у оквиру SCADA HMI апликације
3. Успостављање аутоматског управљања
4. Напад на систем
5. Детекција напада путем машинског учења

### Имплементација Modbus протокола и дефинисање система

У оквиру решавања овог дела проблема бавили смо се самим Modbus протоколом и ушли у срж његове дефиниције како би извршили што бољу и веродостојнију имплементацију [1][2].



Слика 1.0 Формат Modbus порука

На слици 1.0 приказан је формат Modbus порука, а у даљем наставку текста проћи ћемо кроз сам рад протокола и кад се која порука шаље.

У самој основи када посматрамо Modbus поруке имамо два основна типа порука, а то су Write и Read. Ови типови порука имају своје “подврсте” које се дефинишу у зависности од тога какав сигнал желимо да прочитамо или да извршимо упис (променимо његове стање).

Разликујемо 4 типа сигнала:

1. Аналогни улаз - не подржава write
2. Аналогни излаз - подржава read и write
3. Дигитални улаз - не подржава write
4. Дигитални излаз - подржава read и write

Разлози зашто је онемогућено уписивање вредности над аналогним и дигиталним улазом је то што то они представљају улазе у наш SCADA систем и није могуће извршити операцију уписа тј. промене стања система тек тако.

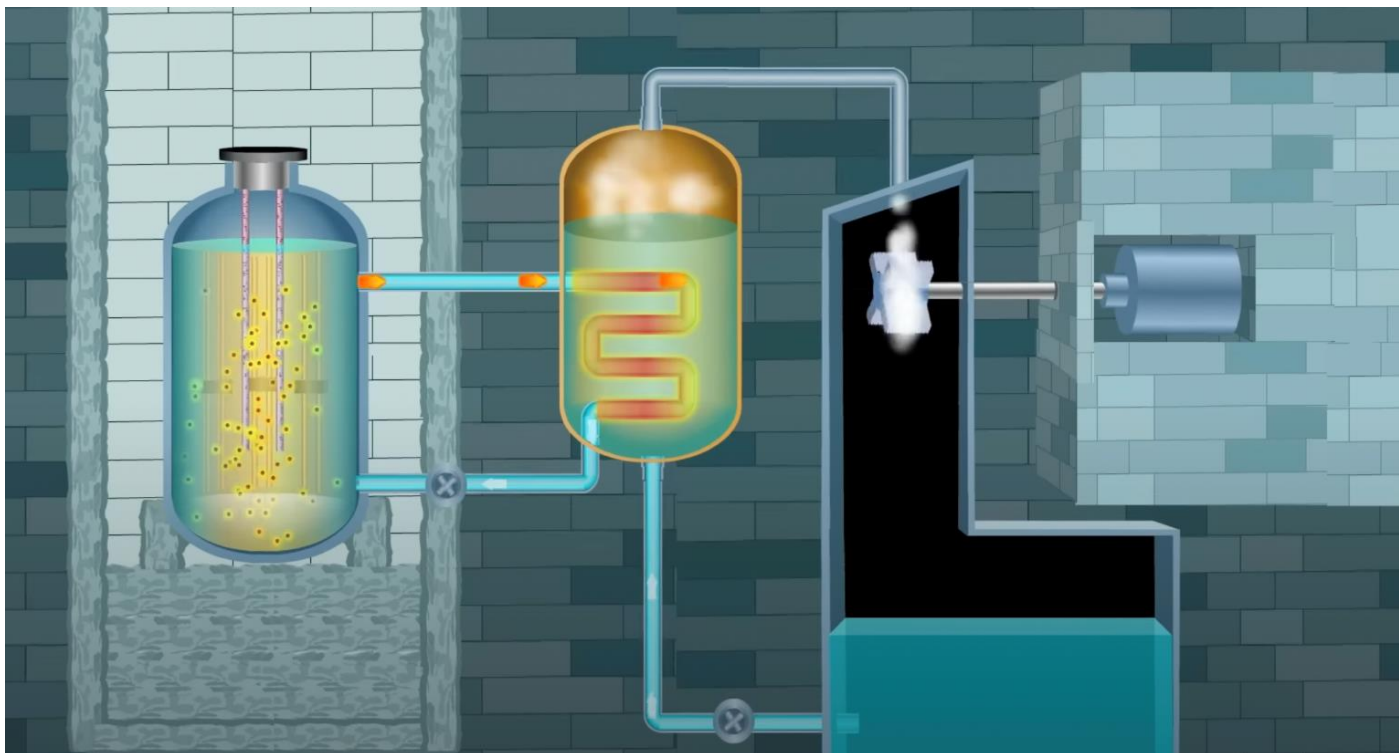
Примери:

Карикирамо као да имамо неки сензор у оквиру нашег система који треба да нам јави да ли је вода попунила наш резервоар до одређене висине. Он треба да јави нашем скада оператеру да ли је вода достигла одређени ниво и сасвим природно је да ми не можемо да вршимо манипулацију над њим пошто нам он даје информације о нашем систему (како је његово тренутно стање) тј. наш сензор ће одреаговати на понашање система.

У другом случају пак кад имамо неки мерач температуре воде он представља наш аналогни улаз и исто тако је природно да ми не можемо да вршимо манипулацију над њим пошто ће он сходно температури воде да мења своје стање које је из неког одређеног опсега (нпр. вредности између 30 и 40 степени).

У оквиру нашег конкретног система имамо један актуатор за контролне решетке које представљају аналогни излаз пошто је над њима могућа манипулација са SCADA HMI као и један аналогни улаз који представља мерач температуре воде. У зависности од температуре воде, контролним решеткама се задаје команда за спуштање/вађење из воде.

У оквиру слике 1.1 налази се сликовити приказ нашег система чију смо симулацију извршили. Конкретно нуклеарна електрана се састоји од два дела система док смо ми моделовали леви део система у оквиру ког се могу приметити контролне решетке (Control Rods).



Слика 1.1. пример система

Након што смо установили шта је наш систем и како ће изгледати као и над којим сигнаlima је могуће вршити какве операције кренули смо у саму имплементацију протокола.

Са слике 1.0 видимо да имамо базну класу `ModbusCommandParameters` у оквиру које се налазе поља које су заједничка свим типовима порука.

#### **ModbusCommandParameters:** [1][2][3]

1. `TransactionID` - ово поље је величине 2 бајта (16 бита) и оно нам само говори који је редни број те наше трансакције
2. `ProtocolID` – говори који протокол користимо. У нашем случају ће ова вредност увек бити 0, због различитих верзија Modbus протокола овај број може да се разликује
3. `Length` – представља величину коју заузима `UnitID`, `FunctionCode` и остатак поруке у зависности од тога да ли је у питању операција читања или писања. Пошто су `UnitID` и `FunctionCode` 2 бајта на ово увек само додајемо величину остатка наше поруке
4. `UnitID` – представља број станице са које вршимо прикупљање података. У нашем случају ово је 100. Овај број нам је доста значајан поготово уколико прикупљамо и приказујемо податке о више различитих SCADA станица на једном месту. Величине је једног бајта (8 бит-а)
5. `FunctionCode` – говори нам која операција се извршава, величине је једног бајта.

#### **Вредности FunctionCode-a:** [1][2][3]

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

Даље као што је горе напоменуто ову класу наслеђују све остале у оквиру Modbus порука.

Тако да када вршимо **ModbusReadRequest** операцију додајемо 2 додатна параметра, а то су:

1. `StartAdress` – величине 2 бајта којим ми дефинишемо са које почетне адресе желимо да прочитамо нешто у оквиру нашег система
2. `Quantity` – исто величине два бајта који говори са колико узастопних локација желимо да извршимо читање тј. на колико узастопних локација се налази наш податак.

На овако послату поруку добијамо одговор који називамо `ModbusReadResponse` који треба да садржи апсолутно исте послате вредности `TransactionID-a`, `ProtocolID-a` и `UnitID-a` док се `FunctionCode.Length` мења због остатка одговора.

#### **ModbusReadResponse:** [1][2][3]

1. `ByteCount` – представља колико бајтова је наш одговор, величине је једног бајта
2. `Data` – представља прочитану вредност тј. вредност коју смо желели да прочитамо када смо послали `ModbusReadRequest`. Величина варира у зависности од тога да ли читамо аналогне или дигиталне вредности, као и колико смо регистара прочитали.

#### **ModbusWriteRequest:** [1][2][3]

Садржи иста поља као и `ModbusReadRequest` што се тиче базне класе а једина поља која

представљају разлику:

1. RegisterAddress – Величине је 2 бајта и представља адресу у оквиру нашег система (у нашем случају адресу у оквиру симулатора) на коју желимо да упишемо вредност
2. RegisterValue – Такође је величине 2 бајта и представља ту вредност коју желимо да упишемо на адреси коју смо претходно навели.

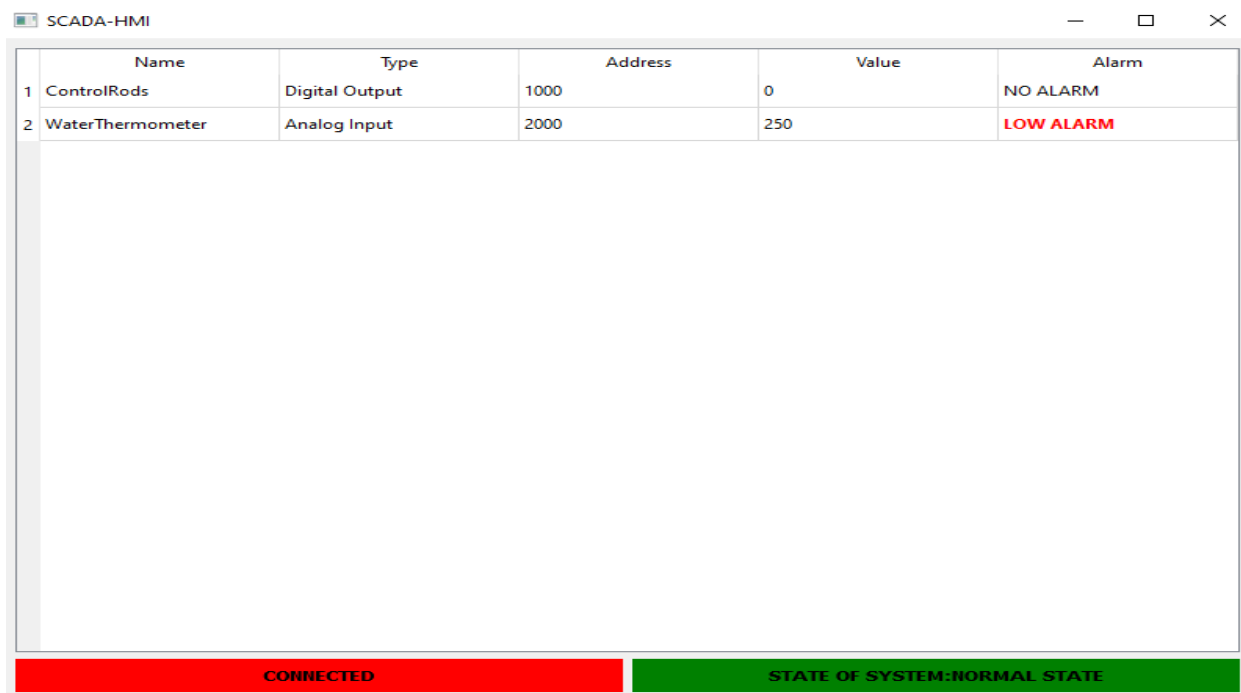
### **ModbusWriteResponse:**

Ова порука је у потпуности иста као и ModbusWriteRequest и уколико нам стигне нешто супротно од нашег скада система значи да се наш захтев за уписом нове вредности није применио.

Када смо дефинисали и успешно извршили комуникацију између симулатора и наше SCADA HMI станице дефинисали смо и процес аквизиције података, који ће констатно извршавати прикупљање нових података. Поруке за читање су фиксне и оне се константно шаљу и наш оператер има увид у најновије стање система.

### **Развој корисничног интерфејса у оквиру SCADA HMI апликације**

Како бисмо имали увид у то како наш систем ради и како би знали да ли је конектован са нашим симулатором изградили смо једноставан кориснички интерфејс како би то могло и да се испрати на слици 2.1 је приказан SCADA HMI када није конектован на симулатор док је на слици 2.2 приказан SCADA HMI када је конектован на симулатор.



	Name	Type	Address	Value	Alarm
1	ControlRods	Digital Output	1000	0	NO ALARM
2	WaterThermometer	Analog Input	2000	250	LOW ALARM

CONNECTED

STATE OF SYSTEM: NORMAL STATE

Слика 2.1

У оквиру интерфејса можемо да видимо неке додатне информације као што су:

1. Name – представља име нашег сигнала интерно у оквиру апликације како би сам оператер знао шта чита заправо
2. Type – представља који је тип сигнала у питању
3. Address – представља адресу на којој се налази сигнал
4. Value – представља тренутну вредност тог сигнала
5. Alarm – представља упозорење да ли је наша вредност дошла у “критичну зону”. Ове вредности се дефинишу само за аналогне сигнале пошто дигитални сигнали сами по себи имају два стања 0 или 1.



У оквиру доњег дела апликације видимо лабелу која нам говори да ли је SCADA HMI конектован на наш симулатор. Уколико је она црвена као на слици 2.1 онда није конектована док ако је као на слици 2.2 зелена онда су апликације конектоване.



The screenshot shows a window titled "SCADA-HMI" with a table of system components. The table has five columns: Name, Type, Address, Value, and Alarm. There are two rows of data. The first row shows "ControlRods" as a "Digital Output" at address "1000" with a value of "0" and an alarm status of "NO ALARM". The second row shows "WaterThermometer" as an "Analog Input" at address "2000" with a value of "250" and an alarm status of "LOW ALARM" in red text. Below the table, there are two green status bars: "CONNECTED" on the left and "STATE OF SYSTEM: NORMAL STATE" on the right.

	Name	Type	Address	Value	Alarm
1	ControlRods	Digital Output	1000	0	NO ALARM
2	WaterThermometer	Analog Input	2000	250	LOW ALARM

CONNECTED STATE OF SYSTEM: NORMAL STATE

Слика 2.2

Ово је одрађено из разлога што се саме апликације могу покретати независно једна од друге. Сам симулатор може да врши симулацију рада система без да је конектован на SCADA HMI као што би имали заправо у реалним условима ако дође до неких сметњи наш оператер не би имао увид у рад система али он би наставио да ради.

#### Успостављање аутоматског управљања: [1]

Како наш оператер не би морао констатно да врши неко управљање у оквиру система дефинисали смо аутоматско управљање приликом чега ће сама апликација у зависности од прикупљених вредности да одреагује и пошаље команде, самим тим спречи систем да се понаша непожељно.

Због самог аутоматског управљања морали смо да дефинишемо и неке границе као што су low alarm и high alarm.

Наш систем симулира језгро нуклеарног реактора који се састоји од резервоара са водом у којој се налазе решетке уранијума који испуштају неутроне који греју воду. У резервоар се могу спустити додатне контролне решетке које успоравају нуклеарну реакцију и хладе систем. Када оне нису у води (подигнуте су), реакција се убрзава и температура воде се повећава.

Дакле, контролне решетке могу бити у стању 0 (подигнути) или 1 (спуштени).

Док термометар воде мери температуру од 0 до 700 при чему се high alarm пали на 350 док се low alarm пали на 250. Из овога можемо закључити да је нормално стање система када је температура коју читамо са термометра воде >250 или <350.

#### Напад на систем [5]:

Приликом самог напада на систем смо морали да се ставимо у кожу нападача и да радимо неке од поступака које он такође ради.



### Кораци:

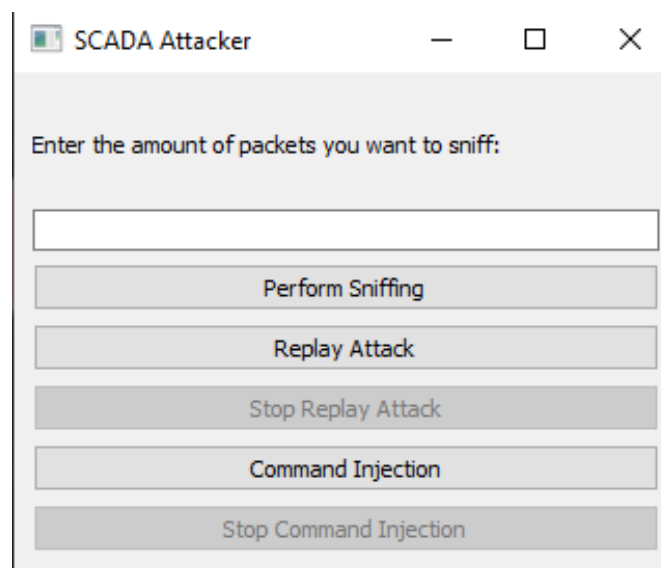
1. Нападач прво треба да изврши прикупљање пакета који се размењују између два елемента SCADA система, у нашем случају Master-а (SCADA HMI) и Slave-а (Симулатора Постројења).
2. Након овога следи само анализирање пакета и упознавање са протоколом. У оквиру овог корака нападач се упознаје са самим протоколом у овом случају Modbus и гледа његове недостатке и начине како да искористи те недостатке како би нарушио само функционисање система.
3. У овом кораку нападач врши само нападање на систем у шта спада прикупљање пакета и њихову измену/ретрансмисивање. Овај корак смо извршили помоћу PyDivert wrapper-а за WinDivert C библиотеку која омогућава пресретање пакета на мрежи из user-space-а. На слици 3.1 приказан је начин како се уз помоћ библиотеке врши хватање пакета који има одредишни порт 1234 и изворни порт 80. [5]

```
import pydivert

with pydivert.WinDivert("tcp.DstPort == 1234 or tcp.SrcPort == 80") as w:
    for packet in w:
        if packet.dst_port == 1234:
            print(">") # packet to the server
            packet.dst_port = 80
        if packet.src_port == 80:
            print("<") # reply from the server
            packet.src_port = 1234
        w.send(packet)
```

Слика 3.1

Напад се састоји из 2 корака од којих је један опциони:



Слика 3.2

На слици 3.2 приказан је кориснички интерфејс преко ког се врши нападање.

Корак њушења пакета је овде опциони из разлога што када извршимо једном “њушење” (прикупљање пакета), њих можемо стално да користимо како би вршили нападе.

Уколико би укуцали одређену вредност нпр. 100, толико би пакета било прикупљено након притиска на дугме. Док остали дугмићи покрећу Replay Attack тј. Command Injection или га заустављају.

Replay Attack је реализован тако што се вршила ретрансмисија пакета у дискретним временским тренутцима и за њега смо користили пакете које смо прикупљали. Ретрансмисија пакета се вршила тако што смо слали 5 пакета који су раније прикупљени док је сваки 6. пакет био заправо прави

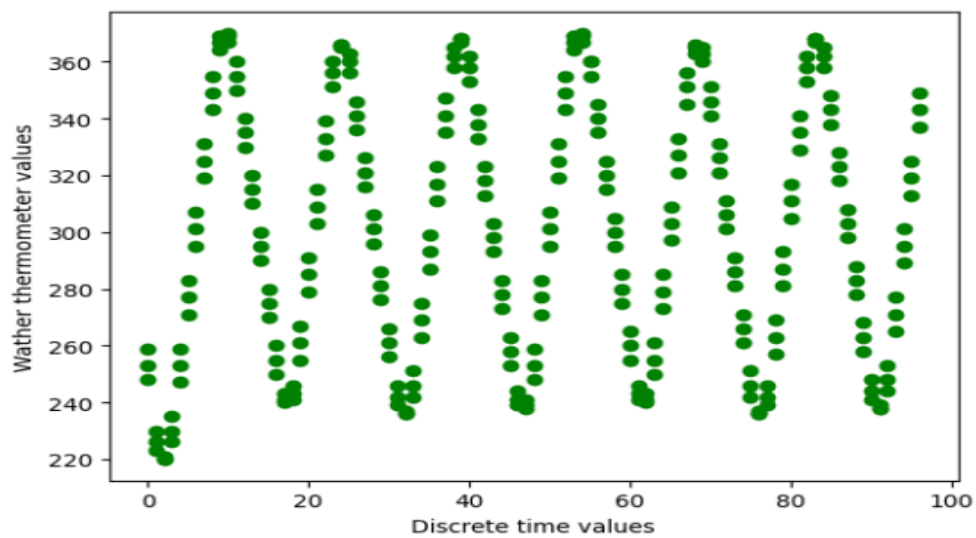
податак који је и требао да стиже на SCADA HMI.

Док је Command Injection реализован тако што је вршено уметање команде приликом њеног задавања (приликом слања са SCADA HMI на симулатор), и касније у повратку је извршена преправка пакета како би SCADA HMI мислио да је његова команда успешно извршена док је сасвим друго стање на симулатору. Неопходно је било извршити преправку пакета при повратку на ону вредност која је послата са SCADA HMI-а пошто би се приказало право стање на SCADA HMI, те би било лакше уочљиво да је дошло до неког пропуста у трансмитовању поруке, тј. Write Request је измењен. Овим смо постигли да нам се температура пење до максимума и да је цело аутоматско управљање онеспособљено.

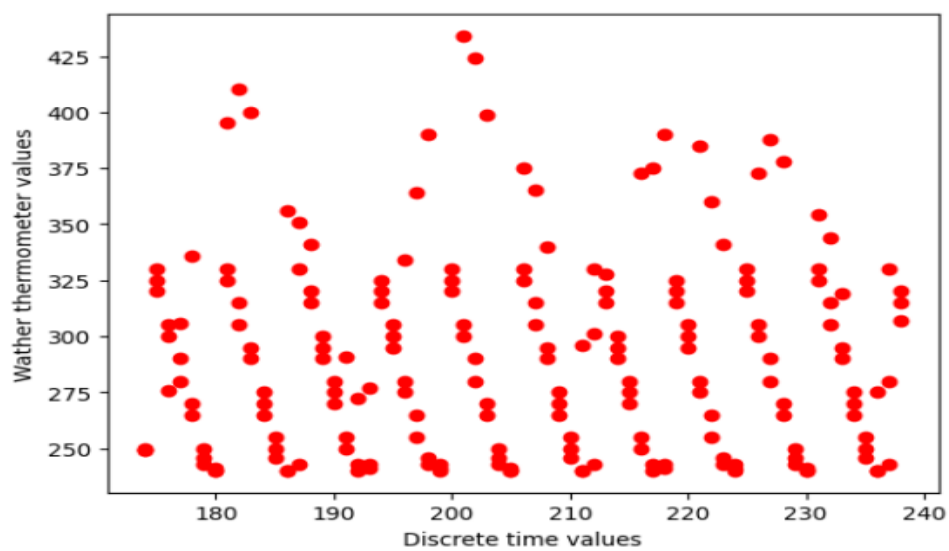
На слици 3.3 је приказан рад система када нема никаквог напада када систем ради у нормалном стању.

На слици 3.4 је приказан рад система приликом Replay attack-а.

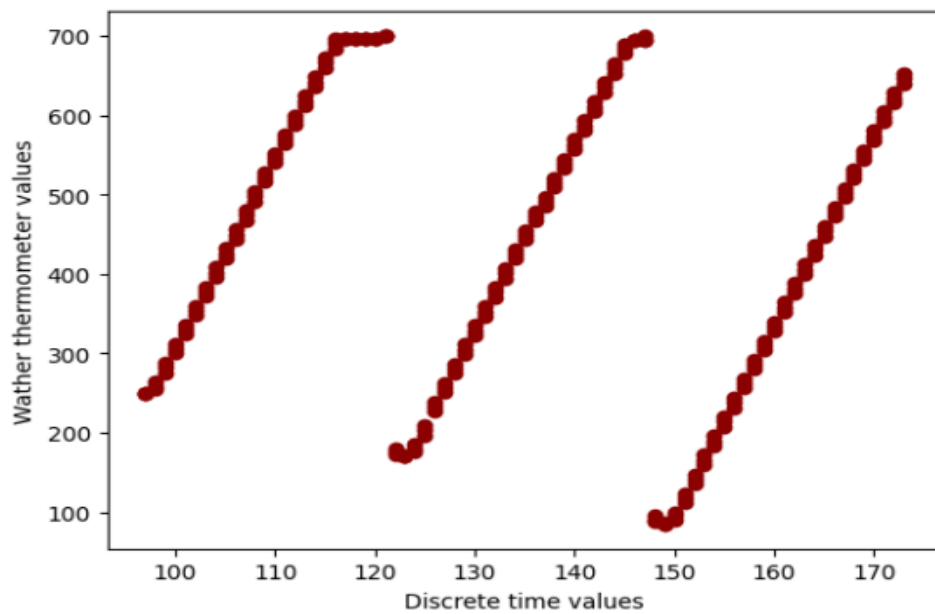
На слици 3.5 је приказан рад система приликом Command Injection attack-а.



Слика 3.3 систем у нормалном режиму рада



Слика 3.4 систем приликом Replay attack-a.



Слика 3.5 стање система приликом Command Injection-a

## Детекција напада путем машинског учења

За детекцију напада смо користили данас веома популарну технику машинског учења познатију као Decision tree тј. Random forest algorithm. [6][7]

Random forest algorithm псеудокод: [6][7]

*Given training set of size  $m$*

*For  $b=1$  to  $m$ :*

*Use sampling with replacement to create a new training set of size  $m$*

*But instead of picking from all examples with equal ( $1/m$ ) probability, make it more likely to pick examples that the previously trained trees misclassify*

*Train a decision tree on the new dataset*

Разлог зашто смо се одлучили да применимо ову технику а не вештачке неуронске мреже је тај што би мрежа била доста спорија приликом класификације. Наиме decision tree ће се много боље понашати када има структуриране податке, док код неструктурираних података као што је аудио, видео, слике итд. мрежа се понаша доста боље. Разлози зашто неки тимови у свету данас имају доста боље и брже моделе је разлог зато што користе decision tree који брзо врши одлуке у односу на неуронску мрежу.

Код конкретног нашег проблема вршили смо бележење података када је систем у нормалном стању рада и када су извршени напади, тако су и настали дијаграми са слика 3.3, 3.4 ,3.5

### Процес тренирања и прављења модела:

```
boost = np.loadtxt("learningDataNew.csv",delimiter=',',skiprows=1)
#preskacemo prvu vrstu zato sto nam je to header
X = boost[:, 0:6] #prvih 6 kolona su nam zapravo input
Y = boost[:, 6:9] #poslednje 3 kolone su targeti

X_train, X_test, y_train, y_test =
sklearn.model_selection.train_test_split(X,Y,test_size=0.35,random_state=39)

model = xgb.XGBClassifier(random_state=42, max_depth=9, tree_method="hist",
multi_strategy="multi_output_tree",n_estimators=100,nthread=5,learning_rate=0.32)
print(model)
```

Слика 4.1 креирање модела

### **TRAIN\_TEST\_SPLIT [8]**

Величина тестног скупа је препоручљива да буде између 0.2-0.35, ако би тестни скуп био превише мали модел би се на око чинио добар (нпр test\_size=0.1), али када би добио нове податке скроз би се изгубио (не би знао модел добро да предиктује).

random\_state - Дефинише начин на који ће се вршити подела самог dataset-а ако нисмо дефинисали увек ће другачија подела бити извршена, док ако ставимо неки целобројни податак увек ће се вршити иста подела. Експериментално приликом развоја модела 39 се показала као најбоља одлука.

## Креирање модела [8]

Random state - ради на сличном принципу као и код train test split-а само се не формира скуп за тренирање него decision trees, исто је експериментално утврђено да је 42 најбоља опција.

**MAX\_DEPTH** - дефинише максималну дубину стабла како би се спречио overfitting, ако ми не дефинишемо по default-у је max\_depth = 10. Експериментално дошли смо до закључка да дубина 9 даје најбољи модел 84+% тачности, док дубина 6 даје 82+%. Поставља се питање да ли узети веће стабло са већом тачношћу или је ипак боље узети мање стабло где брже долазимо до одлуке. Одговор лежи у томе да у системима који су учени на већој количини података 2% нам не значе претерано ако долазимо брже до одлуке. За наш систем смо користили веће стабло пошто је систем учен на малој количини података и свакако нам модел доста брзо одлучује, једноставно смо хтели већу прецизност.

**TREE\_METHOD** - Са овим дефинишемо начин оптимизације формирања нових стабала. Када смо овом параметру доделили hist он ће користити greedy algorithm како би брже формирао нова стабла на основу историје већ креираних.

**MULTI\_STRATEGY** - Овај параметар се користи у комбинацији са three\_method="hist" (препоручено је). Говори да ће број листова бити једна са бројем output-а који треба да се предиктује.

**N\_ESTIMATORS** - Заправо говори колико ће бити В из псеудокода. Експериментално се 100 показало као најбоље.

**NTHREAD** - Колико ће тредова да врши ту обраду. Експериментално се 5 показало као најбоље.

**LEARNING\_RATE** - Колико ће брзо да учи модел посто ипак сам назив је EXTREME GRADIENT BOOST (скраћеница од xgboost). Експериментално се 0.32 показало као најбоље.

```
model.fit(X_train,y_train) #treniranje modela
pred = model.predict(X_test) #predikcija nad testnim podacima
print(pred)
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.] |
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 ]
```

Слика 4.2 тренирање модела

Принцип вршења предикције:

[1 0 0] - Десио се Replay Attack

[0 1 0] - Десио се Command Injection Attack

[0 0 1] - Систем је у нормалном стању рада без напада

Нашем SCADA оператеру неће овако бити представљане ове вредности него ће он при пристизању сваке вредности добијати лог на терминалу који ће му индиковати на то какво је стање система.

```
acc = sklearn.metrics.accuracy_score(y_test, pred) #racunanje preciznosti modela
print("Acc: %.2f%%" % (acc * 100.0))
print(f'Mean squared error:{sklearn.metrics.mean_squared_error(y_test, pred)}')
```

Acc: 84.52%

Mean squared error:0.09126984126984128

Kako ne bi svaki put vrsili ponovno treniranje ovako natreniran model sacuvacemo i kasnije samo ucitati za upotrebu.

```
model.save_model('xgbNew.json')
```

```
newModel = xgb.XGBClassifier()
newModel.load_model('xgbNew.json')
print(newModel)
```

Слика 4.3 прецизност и чување модела

## ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

### 1. Интеграција додатних сензора:

- Треба размислити о укључивању додатних сензора који би пратили додатне параметре, као што су влажност ваздуха, притисак или ниво радијације. Ово би обогатило сет података и обезбедило широку слику стања у постројењу.

### 2. Усмеравање на енергетску ефикасност:

- Развити стратегије за енергетску ефикасност, разматрајући могућности оптимизације употребе енергије у систему. На пример, може се увести алгоритам за управљање потрошње електричне енергије.

### 3. Повећање безбедности:

- Увести двоструку аутентикацију за приступ систему и унапређене енкриптоване методе комуникације. Ово би појачало слој безбедности и спречило недозвољени приступ.

### 4. Развој мобилне апликације:

- Развој мобилне апликације која би омогућила оператерима да прате и контролишу систем на даљини. Ово би било корисно за надгледање система изван постројења.

### 5. Прилагођавање алгоритма аутоматизације:

- Прегледати алгоритме аутоматизације и разматрати опције за њихову оптимизацију. Можда постоје начини за динамичко прилагођавање алгоритма у складу са променама услова рада.

### 6. Развој разноврсних сценарија тестирања:

- Увести разноврсне сценарије тестирања који обухватају различите услове и изазове, укључујући екстремне температуре и варијације околних услова.

### 7. Анализа података коришћењем визуализације:

- Развити механизам за визуализацију и анализу сакупљених података. Графичке представе и облици обраде података могу бити вредан алат за брзе и интуитивне увиде.

### 8. Увођење технологије реалног времена:

- Разматрати увођење технологије реалног времена која би обезбедила још брже и прецизније реакције система на догађаје и команде.

### 9. Сарадња са тимом за безбедност:

- Усмерити пажњу на сарадњу са тимовима за информациону безбедност или експертима за безбедност, како би се обезбедила постојана заштита од потенцијалних безбедносних угрожавања.

## ЛИТЕРАТУРА

- [1] Софтвер са критичним одзивом – пројектовање SCADA система, Бранислав Атлагић 2015
- [2] Modbus Application Protocol specification,  
[https://www.modbus.org/docs/Modbus\\_Application\\_Protocol\\_V1\\_1b3.pdf](https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf)
- [3] Modbus Messaging Implementation,  
[https://www.modbus.org/docs/Modbus\\_Messaging\\_Implementation\\_Guide\\_V1\\_0b.pdf](https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf)
- [4] WinDivert, <https://reqrypt.org/windivert.html>
- [5] PyDivert API Reference, <https://pythonhosted.org/pydivert/>
- [6] Machine Learning Specialization Stanford,  
[https://cs229.stanford.edu/notes2021spring/notes2021spring/Decision\\_Trees\\_CS229.pdf](https://cs229.stanford.edu/notes2021spring/notes2021spring/Decision_Trees_CS229.pdf)
- [7] Decision Tree Model, Andrew Ng Stanford,  
[https://github.com/dusvn/Machine-Learning-Specialization/blob/main/Advanced%20Learning%20Algorithms/C2\\_W4.pdf](https://github.com/dusvn/Machine-Learning-Specialization/blob/main/Advanced%20Learning%20Algorithms/C2_W4.pdf)
- [8] XGBoost Documentation, <https://xgboost.readthedocs.io/en/stable/>