SCHNEIDER ELECTRIC
Нови Сад

Момчило Мицић

Attack on SCADA system

ПРОЈЕКАТ

Нови Сад 28.8.2025.

САДРЖАЈ:

Опис проблема

Циљ рефакторисања пројекта Напади на SCADA систем био је модернизовање програмског кода legacy апликације, побољшавање тестабилности апликације и самим тим омогућавање даљег развоја. Пројекат Напади на SCADA систем био је рађен као супер-пројекат за предмет на факултету и био је наш (мој и мог колеге) први сусрет са било чим напреднијим у Python програмском језику и први покушај комуникације преко мреже две апликације написане на потпуно различитим програмским језицима (C# и Python).
Пројекат се састоји из три апликације:
- Симулатор нуклеарног постројења (аналогни термометар воде која служи као посредник у реакцији и контролни штапови који могу бити подигнути или спуштени и тиме контролишу температуру воде);
- SCADA станица - апликација за надзор и управљање постројењем, приказује врдности сензора и актуатора и у складу са задатом логиком управља постројењем. Садржи и machine learning компоненту која прати стање система и детектује нападе типа Replay Attack и Command Injection;
- Мини апликација која представља кориснички интерфејс за вирус који врши нападе.

Циљ праксе био је рефакторисање саме апликације SCADA станице јер је она најкомплекснија компонента, најлошије је написана. Класе често не постоје, већина функција и променљивих су глобалне, програмске нити нису добро организоване, покрећу се на различитим местима, функције су дугачке, непрегледне и нечитљиве, називи променљивих неинтуитивни. Тестови не постоје. То су само неки од проблема ове апликације које смо покушали да решимо.

Опис решења проблема

Решавање проблема је подељено на три радне недеље, тематски подељене:
1. Израда Safety-net тестова који ће осигурати да се апликација понаша исто пре и после рефакторисања
2. Почетак рефакторисања
3. Завршетак рефакторисања, побољшање апликације и документовање разултата.
Кренимо онда од прве недеље.

Прва радна недеља

Пошто је програмски код апликације био потпуно хаотичан, утврдили смо да није било смисла да Safety-net тестови буду одрађени преко unit тестова јер они подразумевају да ћемо имати неку основу од које ћемо полазити при рефакторисању. А пошто ћемо ту основу изменити или је негде уопште нисмо имали, најбоље решење је било да Safety-net тестови буду End-to-end, где ће се тестирати коначан приказ критичних варијабли на корисничком интерфејсу. Тако смо тестове поделили на главне Use case-ове:
- рад при нормалној температури,
- рад при повишеној температури,
- рад при сниженој температури.

У сваком од ових случајева смо тестирали:
- приказ температуре на табели,
- приказ аларма,
- логику аутомације,
- приказ статуса конекције са постројењем.

Додатни use case-ови за тестирање натренираног модела машинског учења:
- нормално стање,
- стање Replay Attack,
- стање Command Injection.

Овде смо проверили да ли се показује индикатор напада на корисничком интерфејсу.

Наравно, ни један од ових тестова није могуће извршити без одговарајућег Mock-а постројења. Исти је направљен за сваки тестни случај и шаље и прима одговарајуће податке у односу на потребе тестова.

Ово је било наше прво сусретање са Python тестирањем, PyQt тестирањем (за приступ GUI-ју) и Mock-овањем у Python-у. Такође, сам апликативни код је био толико лоше написан да није омогућавао правилно стартовање и терминацију апликације, па су већ сада морале бити извршене мале промене апликативног кода које су нам онда омогућиле тестирање.

Друга радна недеља

Када смо осигурали основне критеријуме за детектовање нежељених промена понашања апликације, приступили смо рефакторисању.

Прва на реду је била класа CustomWindow која представља главни прозор апликације. У конструктор су стављене све варијабле које ће касније бити коришћене, да се не би касније додавале класи, док су непотребне редом уклањане. Поједностављена је функција init_ui која иницијализује и подешава потребне параметре. Уведене су посебне класе за табелу за приказ стања, лабеле за индикаторе стања конекције и детектованог напада, што је олакшало сређивање методе updateTable која је у неком моменту преименована јер је служила да се периодично ажурира стање целог интерфејса а не само табеле, док је ажурирање појединачних компоненти померено у респективне класе, поштујући Tell don’t ask principle. Након тога смо из истог фајла издвојили функције потребне за покретање и заустављање апликације и сместили их у посебну класу

Application, одакле је било лакше и интуитивније пратити почетни ток апликације. То је такође омогућило обједињавање логике покретања и заустављања апликације приликом тестирања и нормалног коришћења.

Након тога приступило се рефакторисању класе Signal која представља објекат у коме се чувају подаци са регистара са симулатора постројења. Уклоњена су нека поља, већина их је преименована да поштују конвенцију именовања својстава, метода и променљивих у Python-у. Промена аларма је премештена из разних ванских метода у унутрашњост и активира се на саму промену вредности у регистру. У складу са тим, промењена је метода ажурирања табеле која сада сама прави TableRow од Signal-а.

Класа Connection престала је да буде статичка и преименована је у ConnectionHandler и њене функције повезивања, реконектовања, слања и примања порука са и од симулатора постројења су сада thread-safe.

Од фајла DataBase направљена је истоимена класа која служи ономе што му име каже, да складишти тренутно стање регистара, података о постројењу и у једном моменту је служила за држање статуса покренутости апликације које су користиле класе за конекцију и аквизицију и аутомацију. Уједно је срећена и функција load_cfg која у базу учитава конфигурациони фајл са подацима о симулатору и регистрима који треба да постоје у систему.

Сама аквизиција и аутомација су премештене у сопствену класу Executor која сада врши аквизицију, ажурира базу и на основу трнутног стања врши аутомацију. Срећена је обрада изузетака које може да пошаље симулатор.

Класа ModbusBase је сада апстрактна и обједињује заједничка поља свих осталих Modbus класа.

Овиме смо заокружили другу радну недељу где смо главне ствари преименовали ради читљивости разумљивости и боље организације. Увели смо нове класе, стабилизовали понашање апликације и уклонили редундантне променљиве и функције и тиме скратили величину програмског кода.

Трећа радна недеља

За ову недељу имао сам више идеја али због недостатка времена одлучио сам се за само неколико од њих. То су мања промена архитектуре и дизајна апликације, мало побољшање и коначно писање документације.

Хтео сам да учиним да логика апликације буде више loosely coupled, то јест да класе буду независније једна од друге. На пример, хтео сам да избацим да променљиве о стањима класа ConnectionHandler и Machine Learning Model буду унутар њих самих и да оне обавештавају друге класе о својим унутрашњим променама уместо да све то трпају у базу података коју ће касније неко други pool-овати. Тиме се родила идеја за увођењем event-based архитектуре.

Тако сада класа ConnectionHandler обавештава базу о промени статуса конекције и база емитује догађај који остале класе ослушкују. Executor обавештава базу о промени статуса у регистрима и база емитује догађај који ослушкује кориснички интерфејс. MachineLearningModel обавештава о промени детектованог стања. За последицу је то имало увођење класа које ће да представљају стање конекције и детектовано стање

система. Тиме смо омогућили да то стање промени индикаторе о конекцији и стању система, без if-else блокова што олакшава читљивост кода.

Као последица тога, класи CustomWindow уклоњена је функција update_gui јер се база сада не pool-ује него кориснички интерфејс реагује на емитоване догађаје из базе. Начин на који је то изведено специфично за GUI је мало другачији јер је GUI одрађен у PyQt-ју што је C++ библиотека и не реагује добро на промене из нити која шаље евент. Потребно је то напоменути али је и то срећено.

Рефакторисана је и логика аутомације и за њене потребе уведене су нове апстрактне класе ModbusRequest и ModbusResponse које имају своје конструкторе у зависности од потребе.

Остало је одрадити ситније пеглање одређених класа и срећивања откривених багова који нису битно утицали на функционалност апликације. На пример, сада MachineLearningModel не врши проверу стања ако није конектован на постројење. Такође је обезбеђено правилно затварање socket-а и уклоњено потенцијално цурење меморије.

Тиме је уоквирено рефакторисање апликације те сам се бацио на додавање једне битне и занимљиве функције: графикона за приказ недавних вредности регистара. Додати су и тестови за то.

Закључак

Сматрам да је циљ скоро у потпуности остварен:
- програмски код апликације је вишеструко читљивији, поштује неке принципе чистог кода,
- програм поседује safety-net end-to-end тестове,
- програм је стабилнији и тестабилнији,
- додата је нова функционалност и тиме употпуњена сврха SCADA станице.

На личном плау сам такође презадовољан:
- први сусрет са разним врстама тестирања у Python-у, научио сам како треба и како не треба тестирати
- први сусрет са Mock-овањем у Python-у
- научио сам многе ствари о функционисању и принципима писања кода у Python-у
- почетак уплитања у event-based програмску архитектуру је прошао солидно
- ново искуство у раду са конкурентним вишенитним програмирањем
- за три радне недеље сам поред посла и спремања испита успео да одрадим пројекат, у језику који смо мање радили на факултету, линеарним напретком без вишедневног задржавања на неком проблему.

Предлози за даља усавршавања

У недостатку времена било је вагања између писања unit тестова и додавања нове функције али пошто сам претходне три недеље радио само на поправљању нечег већ постојећег, одлучио сам се за додавање нове функционалности програма. Дакле прво што би следеће требало одрадити јесу unit тестови.

Као и увек, увек може боље па тако и овде је могуће даље срећивање апликације, преименовање неких класа (Signal), одвајање базе од event hub-а, додатно

рефакторисање ML класе итд. Како је рекао један претходни ментор, треба слушати шта код има да каже. Тако да је довољно само ући у изворни код и идеје ће кренути саме да се рађају.

Одувек сам имао жељу да променим начин на који је модел машинског учења трениран. Мишљење је да би га требало тренирати на променама вредности уместо на конкретним вредностима у регистрима, тиме би модел инхерентно научио понашање система уместо да памти конкретне вредности.

Апликација симулатора постројења остала је нетакнута и иако је доста мања у поређењу са главном апликацијом надзорне станице, и њој би добро дошли тестови и рефакторисање.