Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 007:004.056
DOI: https://doi.org/10.24867/12BE01Sunjka

ПРОШИРЕЊЕ АЛАТА ЗА АУТОМАТСКУ ДЕТЕКЦИЈУ ПРЕТЊИ СА *CWE* БАЗОМ ЗНАЊА И ЊЕГОВА УПОТРЕБА У РАЗВОЈУ БЕЗБЕДНОГ СОФТВЕРА

EXTENSION OF AN AUTOMATIC THREATS DETECTION TOOL WITH CWE KNOWLEDGE BASE AND ITS USE IN THE DEVELOPMENT OF SECURE SOFTWARE

Јово Шуњка, *Факултет техничких наука, Нови Сад*

Област – СОФТВЕРСКО ИНЖЕЊЕРСТВО И ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ

Кратак садржај – *У овом раду описан је алат за подршку SDL процеса и смернице за његову интеграцију у процес развоја безбедног софтвера.*

Кључне речи: *Претња, рањивост, слабост, SDL, моделовање претњи.*

Abstract – *This paper presents support tool for SDL process and guidelines for its integration into the development process of secure software.*

Keywords: *Threat, vulnerability, weakness, SDL, threat modeling.*

1. УВОД

Људи су одвајкада морали да обезбеде и заштите себе и своје ресурсе које поседују, тако је и у данашње време, само што се све дешава и у реалном и у дигиталном свету. Поред личних (приватних) података за које је неопходно обезбедити приватност, данас постоје и разне компаније које поседују одређене ресурсе или пружају одређене услуге у дигиталном свету, које желе да заштите од сваког вида злоупотребе. Да би се постигао висок степен безбедности, неопходно је приступити обезбеђивању софтверских система на систематичан и стандардизован начин, што спада у домен развоја безбедног софтвера (енгл. *Security Development Lifecycle, SDL*).

У контексту софтвера, *SDL* је процес који проширује животни циклус развоја софтвера (енгл. *Software Development Lifecycle)* тако што сваку његову фазу проширује безбедносним активностима (праксама) [1]. Најједноставније речено, *SDL* помаже инжењерима софтвера да развију безбеднији софтвер смањењем броја и озбиљности рањивости софтвера, истовремено смањујући трошкове развоја [1].

Кроз овај рад је описан алат за подршку *SDL* процеса уз смернице за његову интеграцију у процес развоја безбедног софтвера. Алат представља веб базирану апликацију за управљање рањивостима и слабостима у софтверу, са акцентом на *CWE* (енг. Common Weakness Enumeration) модулу. Наведени модул помаже инжењерима да моделују претње, и дизајнирају, имплементирају и тестирају прикладне безбедносне механизме за посматрани софтвер.

2. *SDL*

SDL (енгл. *Security Development Lifecycle*) је процес који проширује Животни циклус развоја софтвера (енгл. *Software Development Lifecycle*) тако што сваку његову фазу проширује безбедносним активностима (праксама) [1]. Циљ *SDL*-а је да помогне инжењерима софтвера да развију безбеднији софтвер. Један од лидера у индустрији који дефинише и практикује *SDL* јесте *Microsoft*. Колико је *SDL* важан, говори и то да постоје индустријски стандарди (*IEC* [5], *NIST* [6]) који га стандардизују.

2.1. Преглед *SDL* приступа у индустрији

Microsoft дефинише *SDL* [7], као скуп од 16 безбедносних активности (пракси) које се интегришу у животни циклус развоја софтвера. Наведени скуп пракси није једини [7] и SDL процес предвиђа проширивање скупа пракси спрам потреба пројектног тима и предлога саветника за безбедност, како би допринеле повећању безбедности софтвера. Једну од обавезних пракси представља безбедносни тренинг (енгл. *security training*) који треба да претходи осталим праксама, а остале обавезне праксе су груписане по фазама традиционалног животног циклуса развоја софтвера.

Без обзира која методологија развоја софтвера се користи, едукација и тренинзи о безбедности су пресудни. Стога би требало следити стандардну *SDL* политику и најмање једном годишње обучити све инжењере о основним безбедносним питањима.

Фаза захтева састоји се од три обавезне праксе: безбедносни захтеви, капије квалитета и *bug bars*, и процена ризика за безбедност и приватност. У дизајн фази примењују се три обавезне праксе: захтеви за дизајн, смањивање површине напада и моделовање претњи. Фаза имплементације састоји се од три обавезне праксе: коришћење одобрених алата, застареле небезбедне функције и статичка анализа. У фаза верификације примењују се три обавезне праксе: динамичка анализа програма, *fuzz* тестирање и модел претњи и преглед површине за напад. *Release* фаза састоји се од три обавезне праксе: план реаговања на инциденте, завршни преглед безбедности и *release / archive*.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био доц. др Никола Лубурић.

Група *IEC* 62443 стандарда пружа флексибилан оквир за решавање и ублажавање тренутних и будућих безбедносних рањивости у индустријски аутоматизованим и управљачким системима (енгл. *industrial automation and control systems, IACS*). Конкретно, *IEC* 62443-4-1 стандард дефинише захтеве за безбедан развој производа који се користе у *IACS*-у и дефинише *SDL* за развој и одржавање безбедних производа [5]. По *IEC*-у, *SDL* треба да се састоји од следећих пракси: дефиниција безбедносних захтева, безбедан дизајн, безбедна имплементација (укључујући смернице за кодирање), верификација и валидација, *defect management*, *patch management* и крај животног века производа [5]. Ове праксе могу бити примењене на нове или на постојеће процесе за развој.

NIST SP 800-160 публикација говори о перспективи вођеној инжењерингом и радњама неопходним за развој система који ће бити више одбрањиви и одрживи. Идеја је да се безбедносна питања решавају тако што ће се одговорити на потребе и захтеве заинтересованих страна и да се користе проверени (устаљени) инжењерски процеси како би се постигло решавање са одређеним нивоом поузданости и одрживости система током целог животног циклуса [6]. Такође као и *Microsoft* и *IEC*, и *NIST* уводи безбедносне активности и задатке, који су развијени као безбедносна проширења процеса животног циклуса система. Активности и задаци заснивају се на принципима и концептима безбедности и поверења.

2.2. Преглед моделовања претњи

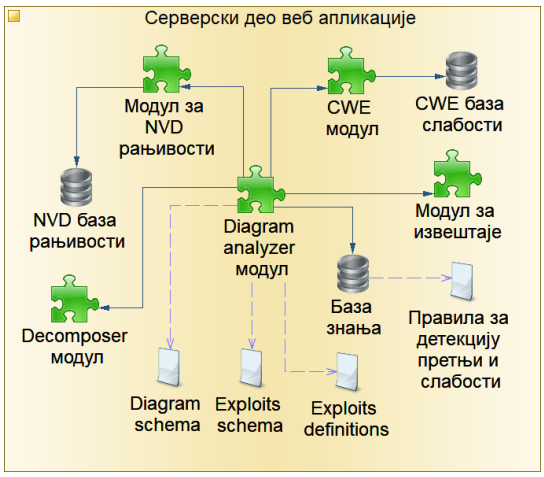
У овом раду, посебна пажња ће бити посвећена моделовању претњи, односно алату за моделовање претњи који је развијен како би се што више олакшало извршавање ове активности.

Моделовање претњи може се извести на више начина. Моделовање базирано на дијаграмима тока података (ДТП) је вероватно најбољи избор, јер често се визуелно могу „изговорити“ ствари које би било тешко објаснити речима, а оне су корисне и за техничку и за нетехничку публику, од програмера до извршног директора. У моделовању претњи могу учествовати чланови тима са различитим улогама. Најчешће улоге које учествују су: *product owner*, архитекта, *developer* и тестер. На сваку од ових улога модел претњи и генерално *SDL* имају значајан утицај. Задатак *product owner*-а је да дефинише потребну безбедност за производ.

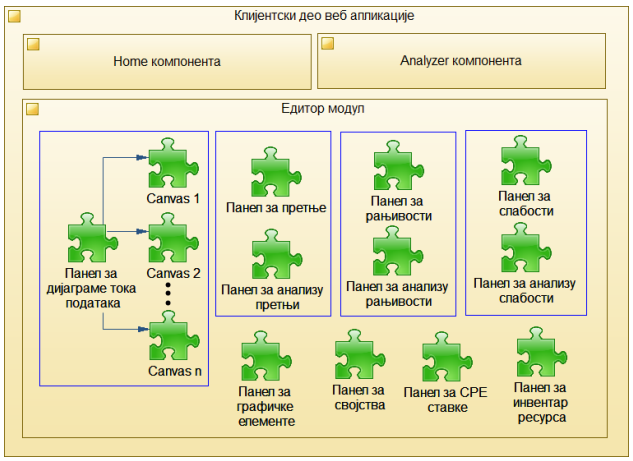
Његове одговорности су да одреди изворе безбедносних захтева и дефинише безбедносне захтеве за развојне тимове. Такође, *product owner* је тај који одлучује да ли ће тренутно већи приоритет имати нпр. развијање неке нове функционалности или решавање неког безбедносног проблема. Софтверски архитекта је задужен да од почетка прави *secure by design* софтвер. Његове одговорности су да: дизајнира вишеслојну одбрану, примењује *secure design* принципе и декомпонује производ за формирање модела претњи. *Secure design* принципи који су написани 1975. године, и данас важе и посебно се односе на безбедносни софтвер [7]. Сваки *developer* да би направио квалитетан софтвер мора да тежи ка писању безбедног кода. Безбедно кодирање се постиже тако што ће се усвојити најбоље праксе безбедног кодирања које су дефинисане у *SDL*-у [7]. У моделу претњи је могуће наћи потенцијалне слабости и рањивости, као и решења за њих. *Developer*-у је објашњено које конфигурације и методе из одговарајућих библиотека треба да користи, као и значење и утицај улазних параметара тих метода на целокупну безбедност софтвера. Да би софтвер био што квалитетнији неопходно га је тестирати, како због проналаска обичних багова, тако и због безбедносних рањивости. Модели претњи су непроцењиви документи које треба користити током тестирања безбедности софтвера. Моделе претњи треба користити за вођење и информисање о плановима за безбедносно тестирање. Такође, најризичнији делови апликације, обично они са највећом површином за нападе и претњама које су највећи ризици, морају се најтемељније тестирати. Кроз тестове је неопходно проверити да ли су исправно ублажене или у потпуности уклоњене слабости и рањивости које су претходно пронађене.

3. ПРЕЗЕНТАЦИЈА ИНТЕГРИСАНОГ АЛАТА

Апликација (алат) је настала интеграцијом две веб апликације: Систем за управљање рањивостима у софтверу (СУРС) [2] и Веб апликација за цртање дијаграма тока података (ВАЦДТП) [3], као и додавањем *CWE* модула. СУРС на улазу прима ДТП, декомпонује га на компоненте (шаблоне), затим анализира шаблоне и на крају креира извештај са потенцијалним нападима на софтвер и листом јавно идентификованих рањивости у софтверу [2]. Треба напоменути да је ова апликација настала ослањајући се *desktop* апликацију - Систем за проналажење напада [9]. ВАЦДТП је направљена са циљем да програмерима и софтверским инжењерима поједностави и олакша активност моделовања претњи користећи стандардне, врло интуитивне графичке елементе за приказивање ентитета из било ког софтверског система [3]. На сликама 1 и 2 може се видети архитектура серверског и клијентског дела интегрисаног алата након интеграције и након свих проширивања функционалности.



Слика 1. *Приказ архитектуре серверског дела апликације*



Слика 2. *Приказ архитектуре клијентског дела апликације*

3.1. Кратак преглед старих функционалности

Diagram analyzer је главни модул у серверском делу апликације. На почетку се врши валидација дијаграма тока података и *exploits* дефиниција по одговарајућим *XML* шемама. Затим се декомпоновање препушта *decomposer* модулу. Резултат декомпоновања су шаблони над којима се примењују правила из базе знања. Применом правила биће пронађени потенцијални напади. Затим се проналажење рањивости препушта модулу за *NVD* рањивости. Шаблони, потенцијални напади и рањивости се прослеђују модулу за извештаје како би креира извештај који ће бити испоручен на клијентски део апликације.

Decomposer служи за рашчлањавање дијаграма тока података на шаблоне, како би се смањила сложеност и извршило ефикасније детектовање и анализа претњи.

Модул за *NVD* (енгл. *National Vulnerability Database*) рањивости има две главне функционалности: скидање рањивости са *NVD*-а и претраживање скинутих рањивости по *CPE*-у (енгл. *Common Platform Enumeration*) производа (хардвер, софтвер).

Модул за извештаје се бави креирањем извештаја за анализиране дијаграме тока података. На почетку извештај буде формиран у *XML* формату, уз помоћ *XSLT* преводи се у *XHTML*, а затим се тај *XHTML* трансформише у *PDF* датотеку. У извештају се налазе пронађене претње, рањивости и слабости, као и предлози безбедносних контрола за њихово регулисање.

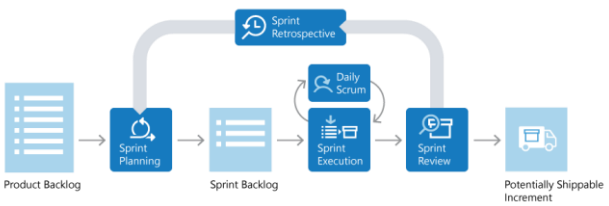
Едитор за дијаграме тока података састоји се од: панела за графичке елементе, панела за дијаграме тока података, *canvas*-а и панел за својства. Панел за графичке елементе садржи графичке елементе који се учитавају из конфигурационе *JSON* датотеке. Дефинисано је пар основних графичких елемената и то су: *process, complex – process, data – flow, external – entity* и *data - store*. Панел за дијаграме тока података је организован кроз табове. *Canvas* је површина на коју се могу превлачити графички елементи и на тај начин се може представити одређени систем. Могуће је изводити стандардне функционалности над графичким елементима. На панелу за својства приказана су својства селектованог графичког елемента и њихове вредности.

3.2. Преглед *CWE* модула

CWE модул има две главне функционалности: скрејповање података о *CWE*-овима и претраживање *CWE*-ова по *id*-ију. Скрејповање података о *CWE*-овима врши се са *cwe.mitre* сајта, а потом ће ти подаци бити кеширани. Треба истаћи да су *potential mitigations, detection methods* и *common consequences* подаци из којих *developer*-и, тестери и остали могу извући много корисних ствари. У *potential mitigations* је објашњено за сваку фазу Животног циклуса развоја софтвера шта се може урадити како би се ублажила одређена слабост система. У *detection methods* се препоручује коришћење разних метода као што су: ручна статичка анализа кода, аутоматска статичка анализа кода, аутоматска динамичка анализа кода, *fuzz* тестирање, како би се на време детектовала одређена слабост система. У *common consequences* се наводе различите последице повезане са слабошћу, идентификују се подручја безбедности апликације која су нарушена, описује се негативни технички утицај који ће настати ако нападач успе да искористи ову слабост, као и вероватноћа да ће се видети конкретна последица у односу на остале последице са списка. Тражење *CWE*-ова проузроковаће база знања, односно правила за детектовање слабости и претњи у систему, у току анализе шаблона екстрахованих из дијаграма тока података. По тренутној имплементацији, у тренутку када *CWE* буде затражен по *id*-ију, прво ће бити проверено да ли се тражени *CWE* налази међу кешираним *CWE*-овима, уколико не буде пронађен, биће извршено скрејповање података за тражени *CWE* са сајта. Уколико тражени CWE буде пронађен у кешу, он се допуњује додатним атрибутима: *diagramName, elementOrFlow* и *severity* (који ће бити израчунат по формули).

4. ИНТЕГРАЦИЈА АЛАТА СА *SDL* ПРАКСАМА У ОКВИРУ АГИЛНОГ РАЗВОЈА СОФТВЕРА

У тренутку када су агилне методологије постале најпопуларније и почеле да се користе у великом броју тимова широм света, појавила се потреба да се *SDL* интегрише са агилним методологијама тако да буду задржани принципи и агилних методологија и *SDL* процеса. Да би *SDL* захтеве (праксе) уклопио у агилни развој, *Microsoft* је сваку праксу из свог класичног *SDL*-а сврстао у једну од три категорије дефинисане по важности за безбедност, а самим тим и по учесталости извршавања тих пракси [9]. У питању су следеће категорије: *every-sprint* праксе, *bucket* праксе (примењују се периодично кроз неколико спринтова) и *one-time* праксе (примењују се на почетку пројекта). У наставку следи пролазак кроз ток рада (енгл. *workflow*) *Scrum* развојног оквира са фокусом на примени алата који је развијен, у циљу реализовања *SDL* пракси (слика 3).



Слика 3. *Приказ тока рада Scrum развојног оквира*

У фази планирања спринта пројектни тим се најпре договара око циља спринта, затим развојни тим одређује ставке из *product backlog*-а које су у складу са циљем спринта, а за које је реално да могу да се реализују у оквиру спринта.

Моделовање претњи је једна од *SDL* пракси која треба да буде примењена за сваки спринт по *SDL*-у за агилни развој. На самом почетку извршавања спринта најбоље би било убацити моделовање претњи које ће се односити само на ставке изабране за овај спринт (енгл. *sprint backlog*). Алат који је описан у овом раду је направљен да олакша моделовање претњи, тако да је ово прави тренутак да буде искоришћен. У едитору алата могуће је скицирати компоненте које се односе на ставке из *sprint backlog*-а, затим те компоненте детаљно описати у панелима за својства и *CPE* ставке, као и кроз панел за ресурсе који се односе на те компоненте, и на крају започети анализу. Алат ће у наставку покушати да детектује претње у систему. Резултат моделовања може бити *export*-овани ДТП који је у току спринта могуће мењати, као и извештај о детектованим претњама, рањивостима и слабостима система.

У току извршавања спринта, било би добро водити се саветима и препорукама које су дефинисане у моделу претњи, било да су генерисане од стране алата или их је нека особа стручна за безбедност унела кроз кориснички интерфејс. Конкретно, током имплементације и тестирања ставки из *sprint backlog*-а корисно би било погледати нпр. секцију за слабости (*CWE*-ове) у моделу претњи, и посебно обратити пажњу на текст који се односи на: потенцијалне митигације, методе детектовања и уобичајене консеквенце.

У току *sprint review*-а особа стручна за безбедност требало би да провери: да ли су *every-sprint* праксе испуњене или су одобрени изузеци за те праксе, да ли је примењена бар једна пракса из сваке категорије *bucket* пракси, да ни за једну *bucket* праксу није прошло дуже од шест месеци а да није примењена, да ниједна *one-time* пракса није премашила рок грејс периода и да нема отворен ниједан безбедносни баг изнад назначеног прага озбиљности (енгл. *severity*). Неки од ових задатака могу захтевати ручни напор стручне особе за безбедност како би се осигурало да је све завршено на задовољавајући начин. Пример једног таквог задатка био би прегледање модела претњи.

У току спринт ретроспективе развојни тим, *ScrumMaster* и *product owner* разматрају да ли у самом *Scrum* процесу који се примењује постоји простора за унапређење. У оквиру ове фазе тим би могао да се консултује и са саветником за безбедност око евентуалних измена на *SDL*-у, тј. око додавања, мењања или избацивања одређених *SDL* пракси, на основу искуства из претходног спринта.

5. ЗАКЉУЧАК

У овом раду описан је алат који представља подршку *SDL* процесу, као и смернице за његову интеграцију у процес развоја безбедног софтвера. Објашњено је шта представља *SDL*, дат је преглед *SDL* пракси и начин на које се оне постижу. Такође, објашњено је како моделовање претњи, као активност, утиче на све учеснике животног циклуса развоја софтвера. У трећем поглављу је презентован интегрисани алат са фокусом на *CWE* модул. Затим је представљено како се алат може интегрисати са *SDL* праксама из другог поглавља о описан је пословни процес употребе алата и *SDL* пракси у оквиру агилне методологије развоја.

Као први предлог за даљи развој ове апликације, предлаже се омогућавање креирања *template*-а на клијентском делу апликације и *upload*-овање *template*-а на серверски део апликације. *Template* би требало да садржи *stencils* (графичке елементе), дефинисане претње, рањивости и слабости. Такође, *template* би могао садржати и правила за детектовање претњи и слабости. Та правила би могла бити описана релативно једноставним домен - специфичним језиком (ДСЈ) који би требало дефинисати. ДСЈ би се преводио у *Drools* правила која се користе и у садашњем систему. На клијентском делу апликације требало би омогућити једноставно креирање правила за детектовање претњи кроз *GUI* (енгл. *Graphical User Interface*), а затим би тај *GUI* био повезан са ДСЈ изразима. Као други предлог, предлаже се детаљно изучавање *CWSS*-а и *CWRAF*-а, као и њихова примена у оквиру ове апликације.

6. ЛИТЕРАТУРА

[1] https://www.microsoft.com/en-us/securityengineering/sdl/practices (приступљено у септембру 2020)

[2] Марија Ковачевић, „Систем за управљање рањивостима у софтверу“, 2019.

[3] Јово Шуњка, „Веб апликација за цртање дијаграма тока података“, 2019.

[4] IEC, International Electrotechnical Commission, „62443-4-1: Security for industrial automation and control systems, part 4-1: Product security development life-cycle requirements“, 2018.

[5] Ron Ross, Michael McEvilley и Janet Carrier Oren, „Nist special publication 800-160: Systems security engineering considerations for a multidisciplinary approach in the engineering of trustworthy secure systems“, 2016.

[6] Michael Howard и Steve Lipner, „The Security Development Lifecycle“, 2006.

[7] Немања Миладиновић, „Проналажење рањивости у софтверу на основу дијаграма тока података“, 2017.

[8] https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ee790620(v=msdn.10) (приступљено у септембру 2020)

Кратка биографија:

Јово Шуњка рођен је 30. новембра 1996. године у Бања Луци, у Републици Српској (БиХ). Основне академске студије на Факултету техничких наука, на студијском програму Софтверско инжењерство и информационе технологије завршио је 2019. године. Тренутно завршава мастер академске студије на истом факултету.
контакт: sunjkajovo@gmail.com