

УНИВЕРЗИТЕТ У БЕОГРАДУ
ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ



АНАЛИЗА АЛАТА ЗА ДЕТЕКЦИЈУ SQLi СИГУРНОСНИХ ПРОПУСТА У АПЛИКАЦИЈАМА

Мастер рад

Ментор:

проф. др Зоран Јовановић, редовни
професор

Кандидат:

Нада Јанковић 3081/2016

Београд, Август 2017.

САДРЖАЈ

САДРЖАЈ	I
1. УВОД	1
2. АНАЛИЗА БЕНЧМАРКА	3
2.1. СИГУРНОСНЕ ПРЕТЊЕ	3
2.1.1. SQL инјекција (SQLi)	5
2.2. ПОКРЕТАЊЕ БЕНЧМАРКА	9
3. АНАЛИЗА АЛАТА	12
3.1. CHECKSTYLE	14
3.2. FINDBUGS и FINDSECBUGS	16
3.3. LAPSE PLUS	18
3.4. PMD	21
3.5. VISUALCODEGREPPER	23
3.6. YASCA	27
4. АНАЛИЗА ТЕСТОВА	30
4.1. ТЕСТОВИ БЕНЧМАРКА ЗА SQL ИНЈЕКЦИЈУ	32
4.2. ИМПЛЕМЕНТАЦИЈА ДОДАТНИХ ТЕСТОВА ЗА SQL ИНЈЕКЦИЈУ	33
4.2.1. Тест „BenchmarkTest02741“	36
4.2.2. Тест „BenchmarkTest02742“	37
4.2.3. Тест „BenchmarkTest02743“	37
4.2.4. Тест „BenchmarkTest02744“	37
4.2.5. Тест „BenchmarkTest02745“	38
4.2.6. Тест „BenchmarkTest02746“	38
4.2.7. Тест „BenchmarkTest02747“	38
4.2.8. Тест „BenchmarkTest02748“	38
4.2.9. Тест „BenchmarkTest02749“	38
4.2.10. Тест „BenchmarkTest02750“	39
5. РЕЗУЛТАТИ	40
5.1. ПРИКАЗ РЕЗУЛТАТА АНАЛИЗЕ АЛАТА КОРИСТЕЋИ ТЕСТОВЕ БЕНЧМАРКА	40
5.2. ПРИКАЗ РЕЗУЛТАТА АНАЛИЗЕ АЛАТА ВЕЗАНЕ ЗА SQL ИНЈЕКЦИЈУ	45
5.2.1. Резултат анализе алата VisualCodeGrepper везан за SQL инјекцију	46
5.2.2. Резултати анализе алата FindBugs и FindBugs са прикључком FindSecBugs везаних за SQL инјекцију	47
5.2.3. Резултат анализе алата Yasca везан за SQL инјекцију	47
5.2.4. Резултат анализе алата Lapse Plus везан за SQL инјекцију	47
5.3. АНАЛИЗА АЛАТА ЗАСНОВАНА НА РЕЗУЛТАТИМА	47
6. ЗАКЉУЧАК	50
ЛИТЕРАТУРА	52
СПИСАК СКРАЋЕНИЦА	54
СПИСАК СЛИКА	55
СПИСАК ТАБЕЛА	56
А. ДОДАТНИ ПРОГРАМСКИ КОДОВИ	57

A.1.	Класа <i>ToXML</i>	57
A.2.	Код класе <i>SEPARATECLASSREQUEST</i> [3]	60
В.	ДОДАТНА ОБРАЗЛОЖЕЊА СИГУРНОСНИХ ПРЕТЊИ	62
B.1.	ИНЈЕКЦИЈА КОМАНДЕ (ЕНГЛ. <i>COMMAND INJECTION</i>).....	62
B.2.	СЛАБ КРИПТОГРАФСКИ АЛГОРИТАМ (ЕНГЛ. <i>WEAK CRYPTOGRAPHY</i>).....	62
B.3.	СЛАБА ХЕШ ФУНКЦИЈА (ЕНГЛ. <i>WEAK HASHING</i>).....	62
B.4.	LDAP ИНЈЕКЦИЈА (ЕНГЛ. <i>LDAP INJECTION</i>).....	62
B.5.	МАНИПУЛАЦИЈА ПУТАЊАМА (ЕНГЛ. <i>PATH TRAVERSAL</i>).....	62
B.6.	КОЛАЧИЋ БЕЗ ПОСТАВЉЕНОГ АТРИБУТА ЗА СИГУРНОСТ (ЕНГЛ. <i>SECURE COOKIE FLAG</i>).....	63
B.7.	КРШЕЊЕ ГРАНИЦА ПОВЕРЕЊА (ЕНГЛ. <i>TRUST BOUNDARY VIOLATION</i>)	63
B.8.	СЛАБ ГЕНЕРАТОР ПСЕУДОСЛУЧАЈНИХ ВРЕДНОСТИ (ЕНГЛ. <i>WEAK RANDOMNESS</i>)	63
B.9.	ХРАТН ИНЈЕКЦИЈА (ЕНГЛ. <i>XPATH INJECTION</i>)	63
B.10.	XSS - ИНЈЕКЦИЈА КЛИЈЕНТСКЕ СКРИПТЕ (ЕНГЛ. <i>CROSS-SITE SCRIPTING</i>)	63
B.11.	РАЗБИЈАЊЕ HTTP ОДГОВОРА НА ДЕЛОВЕ (ЕНГЛ. <i>HTTP RESPONSE SPLITTING</i>).....	64
B.12.	ПРОМЕНА ПАРАМЕТАРА (ЕНГЛ. <i>PARAMETER TAMPERING</i>) И ПРОМЕНА ВЕБ-АДРЕСЕ (ЕНГЛ. <i>URL TAMPERING</i>).....	64

1. Увод

Напретком технологија и алата за развој апликација, захтеви везани за њихово креирање постају динамичнији и самим тим оне пружају подршку за све већи опсег функционалности. Последица овог развоја представља значајно повећање количине података над којима се врше различите обраде. Ово повећање је условило придавање веће пажње заштити апликација од почетка њиховог креирања. У зависности од околности, сигурносни пропусти могу проузроковати незаобилазне проблеме онима који користе апликације. Корисници могу да буду жртве напада у којима се могу открити лични или пословни поверљиви подаци.

Упркос постојању великог броја различитих сигурносних пропусти, њихово проналажење је отежано недостатком познатих ефикасних алата за њихову детекцију. У овом случају се сигурносне претње или занемарују или се примењују алати који у већини ситуација не могу да детектују све претње из дате категорије. Поред недостатка детекције, може се догодити да алат није прецизан, односно да пријављује превише сумњивих претњи које заправо нису претње. Адекватно решење је успостављање метрике за поређење различитих алата, али спровођење овог поступка није једноставно услед постојања различитих фактора који се могу довести у питање попут поменуте прецизности. Поред тога је неопходно дефинисати сигурносне претње и како се спроводе напади на апликације.

OWASP (*The Open Web Application Security Project*) бенчмарк за сигурносну аутоматизацију је најпознатији пројекат који омогућава издвајање ефикасних алата на основу скупа тестова за одговарајућу сигурносну претњу. Захваљујући постојању овог бенчмарка може да се спроведе проучавање предности и мана различитих алата, као и њихово међусобно поређење. У оквиру испитивања описаног у овом раду, примарни фокус је на детекцији SQLi (*Structured Query Language injection*) сигурносних пропусти у апликацијама коришћењем алата за статичку анализу кода, али су анализиране и друге честе претње које се могу занемарити приликом развоја апликација. Поред проучавања алата, извршено је проучавање скупа тестова уграђених у код бенчмарка који су везани за SQLi сигурносну претњу, чиме се додатно истражују начини напада на апликације који их могу компромитовати. Сprovedена анализа тестова је део процеса имплементације нових тестова за детекцију SQLi сигурносних пропусти, који су проширили опсег покривености постојећих тестова.

Истраживање које је спроведено овим радом омогућава детаљнији преглед алата за детекцију сигурносних претњи, који би издвојио најефикасније алате за статичку анализу кода. Као резултат анализе, олакшан је процес избора алата који би могао да се искористи као део процеса развоја апликација који се односи на информациону безбедност. Поред тога, анализирани бенчмарк пружа одређену мотивацију креаторима алата да их направе таквим да пронађу што више дефеката, јер бенчмарк даје визуелни графички приказ резултата поређених алата.

Сам документ је организован у шест поглавља. Након уводног поглавља, у поглављу два овог документа, образложени су детаљи захтева везаних за OWASP бенчмарк, као и сигурносне претње које он покрива. Наведена је и структура апликације у погледу имплементације.

Поглавље три служи за представљање алата за статичку анализу кода који су уграђени у кôд бенчмарка, као и оних који су накнадно додати у оквиру овог рада. Изложене су детаљне поставке сваког алата у оквиру бенчмарка и сигурносне претње које покривају. Додатно је наведен начин детекције претњи сваког алата.

Четврто поглавље демонстрира структуру тестова у оквиру бенчмарка који садрже праве и лажне претње. У овом раду је извршено детаљније проучавање тестова везаних за SQLi сигурносни пропуст, што је резултовало креирањем и додавањем тестова у кôд бенчмарка.

У оквиру петог поглавља, направљен је преглед резултата алата описаних у поглављу три. Резултати се односе на успешност покривености алата за тестове који се налазе у оквиру кода бенчмарка, где су одвојено приказани резултати алата након додавања нових тестова везаних за SQLi сигурносни пропуст описаних у поглављу четири. Направљено је и графичко поређење алата.

У последњем поглављу дати су закључци о извршеном истраживању и предложене су надоградње и идеје везане за коришћене технологије.

2. АНАЛИЗА БЕНЧМАРКА

За утврђивање ефикасности алата за статичку анализу кода, неопходна је употреба система, односно апликације, која би на одговарајући начин издвојила успешне алате за детекцију сигурносних претњи. У таквом систему треба пружити као излаз резултат који ће разликовати алате према атрибутима попут брзине и прецизности. Поред тога, сам процес проучавања различитих алата самостално без употребе система са уграђеним механизмом за издвајање ефикасних алата би био комплексан, где би он представљао велики напор за програмере [1]. Из тог разлога би било добро решење употреба бенчмарка који би конзистентно сакупљао све статистичке податке.

С обзиром на комплексност поменутог процеса, овом тематиком се нису бавиле организације до 2015. године, када је светска непрофитна организација OWASP [2], која се бави проблемима везаним за сигурност софтверских апликација, избацила отворено решење. Она је развила пројекат који врши поређење брзине, покривености и прецизности приликом детекције сигурносних претњи од стране различитих алата. Овај пројекат, који је познат као OWASP бенчмарк за сигурносну аутоматизацију, представља бесплатни систем отвореног кода са пакетом тестова и алата. Његова намена је да прикаже да ли алати исправно врше анализу апликација, што укључује проблеме везане за HTTP (*HyperText Transfer Protocol*) захтеве и одговоре, ток података, управљачки ток, рефлексију, анотације и популарне технологије везане за кориснички интерфејс (доминирају фрејмворци у програмском језику *JavaScript*) [3]. Кôд бенчмарка, укључујући и тестове, написани су у програмском језику *Java*, односно бенчмарк представља *Java Maven* пројекат верзије 1.2. Ово је разлог због којег он тренутно има подршку само за алате који проналазе претње у апликацијама које су имплементирани у програмском језику *Java*.

Поред SAST (*Source code analysis tools/Static Application Security Testing Tools*) статичких алата за анализу кода апликација који се проучавају у овом раду, овај бенчмарк омогућава тестирање два додатна типа алата - динамичких и интерактивних, где се динамички односе на тестирање апликација у стању извршавања, а интерактивни су хибриди статичких и динамичких алата. Интерактивни алати користе агенте покренуте на апликационом серверу који врше анализу апликације у реалном времену, при чему се пружа бољи увид у конфигурацију, ток података, ток управљања, захтеве и везе. С обзиром да постоји већи број SAST алата који су доступни, ово представља разлог за избор овог типа алата у оквиру рада. Алати који су укључени у оквиру бенчмарка, као и алати који су накнадно додати, описани су детаљније у поглављу три.

У наставку овог поглавља, детаљније је образложено које сигурносне претње покрива тренутно актуелна верзија бенчмарка и како се оне детектују. Поред тога, у потпоглављу 2.2., описан је процес инсталације, покретања и имплементационог садржаја бенчмарка.

2.1. Сигурносне претње

Организација OWASP обезбеђује листу са најкритичнијим сигурносним претњама у веб апликацијама, која је позната као „OWASP топ 10“ листа. Она се ажурира у складу са

променама и појавама нових сигурносних претњи, што потврђује њену сврху везану за подизање свести о постојећим ризицима. Поред пружања листе претњи „OWASP топ 10“, наводе се образложења превентивних техника заштите и акција које се могу спровести након њиховог уочавања. На основу ове листе, формирана је листа са једанаест најчешћих претњи које се проверавају у оквиру бенчмарка. У табели 2.1.1. је приказана поменута листа заједно са одговарајућим CWE (*Common Weakness Enumeration*) вредностима, које служе за класификацију претњи, и бројем написаних тестова.

Табела 2.1.1. Сигурносне претње у оквиру бенчмарка

	Назив сигурносне претње	CWE вредност	Број тестова
1	Инјекција команде (енгл. <i>Command Injection</i>)	78	251
2	Слаб криптографски алгоритам (енгл. <i>Weak Cryptography</i>)	327	246
3	Слаба хеш функција (енгл. <i>Weak Hashing</i>)	328	236
4	LDAP инјекција (енгл. <i>LDAP Injection</i>)	90	59
5	Манипулација путањама (енгл. <i>Path Traversal</i>)	22	268
6	Колачић без постављеног атрибута за сигурност (енгл. <i>Secure Cookie Flag</i>)	614	67
7	SQL инјекција (енгл. <i>SQLi - SQL Injection</i>)	89	504
8	Кршење граница поверења (енгл. <i>Trust Boundary Violation</i>)	501	126
9	Слаб генератор псеудослучајних вредности (енгл. <i>Weak Randomness</i>)	330	493
10	XPATН инјекција (енгл. <i>XPATН Injection</i>)	643	35
11	XSS - Инјекција клијентске скрипте (енгл. <i>XSS - Cross-Site Scripting</i>)	79	455

За сваку претњу наведену у табели 2.1.1., формира се скуп тестова, односно сервлети, који садрже појединачне праве или лажне претње. Они не представљају апликације, већ делове кода у програмском језику *Java* који би били издвојени из апликација у реалној ситуацији. На овај начин је покретање тестова бенчмарка једноставније и брже, што не би био случај да су коришћене апликације за тестирање. Поред фајлова тестова написаних у програмском језику *Java*, придружују им се и метаподаци у оквиру XML (*Extensible Markup Language*) фајлова. Преглед структуре и садржаја тестова дат је у поглављу четири.

Из OWASP листе се може издвојити група претњи везана за инјекције као најприсутније претње у апликацијама написаних у програмском језику *Java*. Оне се дешавају приликом слања непроверених података интерпретеру од стране апликација у виду упита или команде. На овај начин се може у потпуности компромитовати апликација модификацијом њених података. Сви фрејмворци овог програмског језика који користе интерпретере или инвокацију

других процеса су рањиви на нападе овог типа, што укључује и компоненте фрејмворка које користе *back-end* интерпретере [4]. Најчешћа претња која је припадник ове групе је SQL (*Structured Query Language*) инјекција, због чега је она у наставку описана и одабрана за детаљније тумачење у овом раду. Остале претње су дефинисане у оквиру прилога В на крају рада.

2.1.1. SQL инјекција (SQLi)

SQL инјекција односно SQLi представља једну од најприсутнијих и најопаснијих претњи које могу утицати на велики број апликација написаних у различитим језицима попут следећих: *Java*, *PHP*, *Python* и многих других. Иако постоје фрејмворци за одређене језике који наводно пружају заштиту од ове претње, треба посветити пажњу на који начин се она пружа, јер у многим ситуацијама она није покривена у потпуности [5].

Напад се извршава тако што нападач може да изврши злонамерне SQL упите намењене бази података који могу довести до неовлашћеног приступа систему, измени података или чак уништавања базе података. Да би се такви упити извршили, неопходно је да нападач има доступну улазну тачку у оквиру апликације преко које ће контролисати изглед динамичког упита који ће се извршити. Слање улазних података апликацији може да се изврши на различите начине, као што су слање кроз поља форме веб странице или као део вредности колачића (енгл. *cookie*), серверских променљивих и веб-адресе [6]. Неовлашћеним приступом систему могу се компромитовати интегритет и тајност, при чему треба издвојити приступ осетљивим подацима, који би оштетио не само апликацију, већ и клијенте који је користе.

На основу претходно наведених последица, може се закључити да уколико апликација пружа начин уноса података од стране корисника без валидације или енковања који ће бити део динамичког упита, онда је она сигурно мета за SQLi. Најчешћи пример рањивости апликације се односи на функционалност пријављивања корисника на систем, где динамички упит може да има форму која је приказана на слици 2.1.1. На њој се може уочити да променљиве *username* и *password* нису проверене ни на који начин пре него што су додате упиту, већ да су директно уграђене у упит који је дефинисан променљивом *sql*. Напад се у овој ситуацији може извршити веома једноставно, јер нападач може да уметне жељени SQL упит у оквиру параметара захтева. Један од начина био би постављањем вредности „*admin' OR 1=1 --*“ за корисничко име и произвољне вредности за лозинку, јер она неће бити део упита, чиме би се исписали сви корисници из табеле *users*.

```
String username = request.getParameter("username");
String password = request.getParameter("password");

String sql = "SELECT * FROM users WHERE username='" + username + "' AND password='" + password + "'";

try {
    java.sql.Statement statement = org.owasp.benchmark.helpers.DatabaseHelper.getSqlStatement();
    statement.execute(sql, java.sql.Statement.RETURN_GENERATED_KEYS);
} catch (java.sql.SQLException e) {
    throw new ServletException(e);
}
```

Слика 2.1.1. Пример рањивости на SQLi

Из разлога што SQLi може да угрози апликацију на различите начине, треба издвојити њену класификацију и уочити на који начин свака класа представља претњу. У зависности од имплементационих могућности, неопходно је одабрати метод заштите који ће осигурати

адекватну сигурност система. Из тог разлога су у наставку у пододељцима дефинисане типичне класе и начини заштите од SQL инјекције.

i) Врсте SQL инјекције

SQLi напади се могу категоризовати у виду различитих класа, где се у зависности од намера нападача они могу извршавати појединачно или у групи. На основу класификације SQLi напада од стране Халфонда, Вијегаса и Орза [6], могу се издвојити следећи типови напада који су највише присутни:

- 1) **Таутологије** су тип напада које се односе на уграђивање кода којим би условни исказ упита био увек тачан. Пример овог напада дефинише се уметањем вредности „*admin' OR 1=1 --*“ за параметар корисничког имена *username* и „*nesto*“ за параметар лозинке *password* у оквиру упита „*SELECT * FROM users WHERE username='* + *username* + *' AND password='* + *password*“, којим се приступа налогу администратора без унете лозинке.
- 2) **Нелегални (логички некоректни) упити** односе се на коришћење порука о грешкама везаним за упите да би се прикупили подаци о бази података за следеће нападе. У случају претходно дефинисаног напада таутологије, порука о синтаксној грешци унетог параметра би била „*Incorrect syntax near 'admin'. Unclosed quotation mark after the character string ' AND password='nesto'.*“
- 3) **Упити са унијом** подразумевају коришћење кључне речи *UNION* да би се злонамерни упит надовезао на намерени упит. Уколико би се у упиту „*SELECT * FROM scores WHERE username ='* + *username*“ за параметар корисничког имена *username* унело „*admin' UNION SELECT * FROM users*“, резултат упита успешног напада би укључио и резултате везане за администраторе и све корисничке налоге дефинисане у систему.
- 4) **Надовезани упити** (енгл. *Piggy-backed Queries*) представљају коришћење граничника упита попут тачке са запетом „*;*“, да би се злонамерни упит надовезао на намерени упит. У случају упита „*SELECT * FROM scores WHERE score='* + *score_value*“, уколико би се за параметар резултата *score_value* унело „*5; DROP TABLE scores*“, извршио би се и покушај брисања табеле са резултатима.
- 5) **Инферентни/Напади закључивања** (енгл. *Inference*) подразумевају прикупљање информација посматрањем понашања апликације на основу датих одговора на затворена питања о бази података, где се могу издвојити слепа инјекција (енгл. *Blind Injection*) и временски напади (енгл. *Timing Attacks*). Пример временског напада над упитом (дефинисаном у примеру напада надовезаних упита) који извршава SQL сервер би користио следећу вредност параметра резултата *score_value* „*5; WAIT FOR DELAY '00:00:10'*“. Уколико је успешан напад, одговор од сервера ће каснити.
- 6) **Алтернативна енковања** се односе на избегавање детекције напада изменом злонамерног упита коришћењем алтернативног енковања попут хексадецималног, *ASCII* и *Unicode*. У случају претходно дефинисаног напада таутологије, уколико би се пружила следећа вредност параметра резултата *score_value* „*5; declare (@a char(20) select @a=0x736875746466776e exec(@a)*“, извршила би се команда *SHUTDOWN*, која је наведена у виду своје хексадецималне репрезентације.

Наведени типови напада се могу груписати у три класе [7-8]. Прву класу чине класични SQLi који се односе на коришћење истог комуникационог канала за извршавање напада и прикупљање резултата. Ова класа, позната као *In-band SQLi*, групише поменуте таутологије, нелегалне упите, упите са унијом, надовезане упите и алтернативна енковања.

Друга класа напада се односи на инферентне нападе тј. нападе закључивања, где нападач не добија директне одговоре приликом напада, него мора да формира одговарајуће претпоставке на основу одговора од апликације и понашања сервера базе података. Њих чине технике слепе инјекције и временских напада. Слепа инјекција се односи на постављање затворених питања у оквиру упита како би нападач извршио анализу, с обзиром да се у овом случају приказује генеричка страница за грешку, а не и сама грешка као код нелегалних упита. Са друге стране, временски напади се воде посматрањем временских кашњења приликом слања одговора од стране сервера базе података, где нападач посматра потребно време за учитавање странице. За разлику од слепе инјекције, код временских напада нападач употребљава условни исказ за који је познато потребно време извршавања попут коришћења кључне речи *WAITFOR*, која за одређено време поставља кашњење одговора од сервера.

Последња класа *Out-of-band SQLi* је супротна *In-band SQLi*, јер нападач не може да користи исти комуникациони канал за извршавање напада и прикупљање резултата. Она зависи од тога које су опције омогућене конфигурацијом на серверу базе података за коришћење од стране апликације, што је чини мање вероватном, јер би се односила на способност сервера да прави DNS (*Domain Name System*) или HTTP захтеве како би нападачу достављала информације [8].

ii) Заштита од SQL инјекције

Избегавање ситуација у којима је апликација рањива од SQLi укључује избегавање коришћења интерпретера. Међутим, уколико се не може избећи њихова употреба, треба користити постојеће сигурне апликационе програмске интерфејсе, који имају уграђене механизме за заштиту попут параметризованих упита и постојећих библиотека за објектно-релационо мапирање као што су *Hibernate* и *Spring* [4]. На основу препорука од стране OWASP организације, примарни начини одбране од SQLi су следећи:

- 1) Употреба пре-компајлиране наредбе (енгл. *Prepared Statement*) коришћењем параметризованих упита;
- 7) Употреба ускладиштених процедура (енгл. *Stored Procedures*) и
- 8) Обрада унетих података од стране корисника ради заштите од специјалних односно небезбедних карактера.

Од наведених препорука, употреба пре-компајлиране наредбе се сматра најбезбеднијом. Коришћење овакве наредбе захтева претходно дефинисање SQL упита, којем ће се касније проследити недостајући параметри или вредности. На овај начин је јасно разграничен код од достављених улазних података. Поред тога, за сваки програмски језик препоручује се коришћење одређених објеката и функција. За програмски језик *Java* се препоручује коришћење следећих апликационих програмских интерфејса и објеката:

- *java.sql.PreparedStatement* објекат из JDBC (*Java Database Connectivity*) апликационог програмског интерфејса, где се пре-компајлиран SQL упит складишти у оквиру овог објекта;
- JPA (*Java Persistence API*), који поред подршке за пре-компајлирани SQL упит пружа могућност коришћења сопственог упитног језика JPQL (*Java Persistence Query Language*);
- *Hibernate* фрејмворк, који поред позиционих параметара као претходна два начина пружа могућност коришћења именованих параметара и
- *MyBatis* фрејмворк, који складишти SQL упите у оквиру XML фајлова [5].

У примеру приказаном на слици 2.1.1., употреба пре-компајлиране наредбе би онемогућила успешност напада, јер би се тражио корисник из табеле *users*, чије се корисничко име поклапа са вредношћу „*admin' OR 1=1 --*“, што је приказано на слици 2.1.2. Иако постоје ретки случајеви у којима се сматра да пре-компајлиране наредбе могу штетити перформансама, треба се усредсредити на заштиту података [2].

```
String username = request.getParameter("username");
String password = request.getParameter("password");

String sql = "SELECT * FROM users WHERE username=? AND password=?";

try {
    java.sql.PreparedStatement statement = connection.prepareStatement(sql);
    statement.setString(1, username);
    statement.setString(2, password);
    statement.execute();
} catch (java.sql.SQLException e) {
    throw new ServletException(e);
}
```

Слика 2.1.2. Употреба *java.sql.PreparedStatement* објекта

Друга два поменута начина одбране се сматрају мање ефикасним у односу на употребу пре-компајлиране наредбе, посебно обрада унетих података од стране корисника ради заштите од специјалних карактера [2]. Међутим, коришћење ускладиштених процедура је за неке стандарде језика подједнако ефикасан механизам одбране, где је разлика између употребе пре-компајлиране наредбе и ње у томе што се SQL кôд за ускладиштену процедуру дефинише и складишти унутар саме базе података, а затим позива из апликације. Пример њене употребе је приказан на слици 2.1.3. који је део бенчмарк теста 52, где је приказана употреба *java.sql.CallableStatement*.

```
String param = scr.getValue("BenchmarkTest00052");

String sql = "{call " + param + "}";

try {
    java.sql.Connection connection = org.owasp.benchmark.helpers.DatabaseHelper.getSqlConnection();
    java.sql.CallableStatement statement = connection.prepareCall( sql, java.sql.ResultSet.TYPE_FORWARD_ONLY,
        java.sql.ResultSet.CONCUR_READ_ONLY, java.sql.ResultSet.CLOSE_CURSORS_AT_COMMIT );
    java.sql.ResultSet rs = statement.executeQuery();
    org.owasp.benchmark.helpers.DatabaseHelper.printResults(rs, sql, response);
} catch (java.sql.SQLException e) {
    if (org.owasp.benchmark.helpers.DatabaseHelper.hideSQLErrors) {
        response.getWriter().println("Error processing request.");
        return;
    }
    else throw new ServletException(e);
}
```

Слика 2.1.3. Употреба ускладиштене процедуре

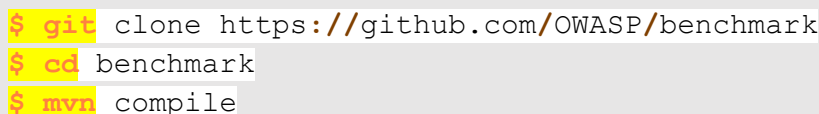
Поред наведених примарних начина одбране, препоручује се примена секундарних додатних начина одбране попут валидације улазних података поређењем са листом валидних улазних вредности (енгл. *Whitelist*) и смањивања привилегија додељених сваком налогу у оквиру базе података (енгл. *Least Privilege*). Први начин од поменута два секундарна може се искористити и као примарни начин одбране, али са опрезом и у случају када није могућа примена других опција [2].

2.2. Покретање бенчмарка

Покретање бенчмарка је битно представити како би се приказала једноставност процеса која омогућава тестирање различитих алата. Наведени поступак инсталације и покретања извршен је на рачунару под оперативним системом *Windows 10*. Предуслови за покретање бенчмарка су претходно конфигурисани:

- алат *Git* [9];
- алат *Maven* верзије 3.2.3 и навише [10] и
- *Java* платформа [11].

Уколико су наведени алати доступни, могуће је започети конфигурацију клонирањем репозиторијума са одговарајуће адресе у оквиру жељеног директоријума на рачунару. Након тога, неопходно је компајлирати изворни код пројекта и покренути га. Описани поступак наведен је на слици 2.2.1., који је базиран на упутствима датим од стране OWASP организације [3], где су команде унете у оквиру *BASH* емулације доступне за покретање алата *Git* из командне линије. Након извршења команди, верзија бенчмарка која је конфигурисана је верзија 1.2.



```
$ git clone https://github.com/OWASP/benchmark
$ cd benchmark
$ mvn compile
```

Слика 2.2.1. Команде за покретање бенчмарка у оквиру *Git Bash*

Структура инсталираног бенчмарка у оквиру директоријума „*benchmark*“ се састоји од следећих битних директоријума:

- „*results*“ - за смештање генерисаних резултата везаних за тестове бенчмарка од стране алата;
- „*scorecard*“ - за смештање генерисаних страница извештаја са резултатима свих алата и сигурносних претњи;
- „*scripts*“ - скрипте за извршавање сваког статичког алата који су већ део бенчмарка (*PMD*, *FindBugs* и *FindBugs* са прикључком *FindSecBugs*) и
- „*src*“ - изворни код бенчмарка укључујући изглед веб страница тестова.

Наведени директоријуми не чине целокупни директоријум бенчмарка, већ су они главне целине за преглед и измену. Поред тога, у кореном директоријуму „*benchmark*“ се налазе следећи кључни фајлови:

- „*expectedresults-1.2.csv*“ - фајл са очекиваним резултатима везаним за тестове. У оквиру њега се наводе информације за све тестове над којима се покреће анализа, односно назив теста, категорија претње, индикатор праве претње и CWE вредност. За тест „*BenchmarkTest00001*“ је у првом реду наведено „*BenchmarkTest00001,pathtraver,true,22*“, што значи да је овај тест садржи праву претњу везану за манипулацију путањама чија је CWE вредност једнака 22.
- „*pom.xml*“ - „*Project Object Model*“ фајл, односно фајл са информацијама о пројекту и конфигурационим детаљима које користи алат *Maven* за подизање пројекта. На

слици 2.2.2. је приказан део садржаја овог фајла, који се односи на основне информације о пројекту и дефинисан профил „*benchmarkscore*“, који је неопходан за генерисање извештаја са резултатима свих алата (поменут у оквиру последње ставке овог набрајања).

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.owasp</groupId>
  <artifactId>benchmark</artifactId>
  <version>1.2</version>
  <packaging>war</packaging>
  <name>OWASP Benchmark Project</name>
  <url>https://www.owasp.org/index.php/Benchmark</url>
  <profiles>
    <profile>
      <id>benchmarkscore</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>1.4.0</version>
            <executions>
              <execution>
                <phase>validate</phase>
                <goals>
                  <goal>java</goal>
                </goals>
                <configuration>
                  <mainClass>org.owasp.benchmark.score.BenchmarkScore</mainClass>
                </configuration>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
```

Слика 2.2.2. Део садржаја фајла „*pom.xml*“

- „*runBenchmark.sh* / *.bat*“ - скрипта за покретање веб апликације бенчмарка. Садржај ове скрипте чини команда „*call mvn clean package cargo:run -Pdeploy*“, којом се покреће фаза испоруке (енгл. *deployment*).
- „*createScorecards.sh* / *.bat*“ - скрипта за генерисање извештаја са резултатима свих алата и сигурносних претњи који се смештају у директоријум „*scorecard*“. Садржај ове скрипте чини команда „*call mvn validate -Pbenchmarkscore -Dexec.args="expectedresults-1.2.csv results"*“, којом се врши валидација пројекта, експлицитно активира профил „*benchmarkscore*“ дефинисан у фајлу „*pom.xml*“ и дефинише својство „*exec.args*“.

Уколико се жели увидети начин извршавања тестова везаних за сигурносне претње у оквиру веб апликације бенчмарка, то се може учинити коришћењем команди у оквиру скрипте „*runBenchmark.bat*“, односно „*runBenchmark.sh*“. Након тога, веб апликацији бенчмарка се може приступити са адресе <https://localhost:8443/benchmark/> по одобравању приступа страници услед појаве упозорења. Изглед почетне странице је дат на слици 2.2.3., где су приказане везе ка страницама са одговарајућим тестовима за сигурносне претње.

Поступци интеграције алата, односно тестова су детаљније приказани у поглављима намењеним за њихову анализу. За интеграцију алата је намењено поглавље три, где је поред

детаљније анализе сваког алата дефинисан начин покретања, док је за интеграцију тестова намењено поглавље четири.

OWASP Benchmark Test Case Index

Available Categories

- [Command Injection Category](#)
- [Cryptographic Category](#)
- [Weak Hashing Category](#)
- [LDAP Injection Category](#)
- [Path Traversal Category](#)
- [Secure Cookie Category](#)
- [SQL Injection Category](#)
- [Trust Boundary Category](#)
- [Weak Randomness Category](#)
- [XPath Injection Category](#)
- [XSS \(Cross-Site Scripting\) Category](#)

Слика 2.2.3. Почетна страница веб апликације бенчмарка

3. Анализа алата

Проналажење дефеката у каснијој фази развоја софтверске апликације може представљати велики проблем по питању ресурса попут времена и финансијских средстава. Због тога тестирање апликације у појединачној или више фаза у току развоја може значајно утицати на решавање дефеката на време, чиме би се избегли додатни ресурси.

Термин статичке анализе кода се односи на аутоматизован процес вршења анализе кода без потребе за његовим извршавањем. На крају обављене анализе, она не може да гарантује да не постоје грешке у фази извршавања. Међутим, постоје одређене грешке које у већини случајева традиционални компајлери не могу да детектују и које се при тестирању тешко проналазе, али се проналазе при статичкој анализи [12]. Примери таквих грешака су: неисправно коришћење ресурса, употреба нелегалних операција, мртав код и подаци, недовршен код, необрађени изузеци, утркивање и слични дефекти. Поред проналажења поменутих дефеката, статичка анализа је корисна као смерница за писање „бољег кода“, проверу типова, дебаговање и генерисање метрика.

Иако се делови кода прогласе као грешке након извршене статичке анализе, дешава се да проглашене грешке заправо нису дефекти. Ова појава доводи до последице повећања непрецизности алата, која је већа уколико је већи број детектованих лажно позитивних исхода. Прецизност може да зависи и од времена потребног за извршавање анализе, где је алат који користи више ресурса прецизнији, што указује на потребно постојање компромиса између ова два фактора.

Према раду који су објавили Емануелсон и Нилсон [12], постоји више различитих начина вршења класификације анализе кода према атрибутима попут управљачког тока, путање и контекста функције, где се свака класификација односи на постојање две класе - осетљиве и неосетљиве на поменуте атрибуте. Код свих различитих подела начина вршења анализе, класе које су осетљиве на атрибуте су прецизније, због чега је потребно више ресурса за њихово извршавање. Ова чињеница указује на претходно поменути компромис који постоји између фактора прецизности и коришћења ресурса.

На основу претходно наведених запажања може да се донесе закључак да је немогуће да алат за статичку анализу кода успева да пронађе све дефекте и да нема ниједан лажно позитиван исход. Алат који проналази све дефекте у коду и има бар један лажно позитиван и ниједан лажно негативан исход дефинише се као коректан (енгл. *sound*) алат [12]. Међутим, проналажење таквог алата је такође тешко, јер се често не пронађу сви дефекти. У поглављу пет приликом приказивања резултата коришћених алата у овом раду, може да се прикаже истинитост наведене констатације.

У оквиру бенчмарка су укључени следећи алати за статичку анализу кода: *PMD*, *FindBugs*, *FindBugs* са прикључком *FindSecBugs* и *SonarQube*. Међутим, није извршено проучавање алата *SonarQube* будући да није било могуће да се репродукује његова анализа због грешке везане за прикључак алата [3]. Ова грешка се примарно односи на некомпатибилност *Maven* прикључка алата, која узрокује прекид конекције и процеса покретања анализе. Поред наведених алата који су укључени у оквиру бенчмарка, придружена су четири алата (*Checkstyle*, *Lapse Plus*, *VisualCodeGrepper* и *Yasca*) која су примарно одабрана

на основу тога што су алати отвореног кода, бесплатни и што се могу интегрисати са бенчмарком. То значи да могу да врше анализу кода написаног у програмском језику *Java*. Како би фајлови са резултатима алата били компатибилни за учитавање од стране бенчмарка, неопходно је да називи фајла буду у следећем формату „*Benchmark_1.2-checkstyle-v3.0.1-1026.xml*“, што се односи на фајл покренут за бенчмарк верзије 1.2 генерисан од стране алата *Checkstyle* верзије 3.0.1 за који је било потребно 1026 секунди за анализу. Иако се фајлови аутоматски генеришу од стране алата, у већини случајева је неопходно ручно преименовати фајл како би био у наведеном формату. Стога је као што је и поменуто у раду, неопходно занемарити назначено време потребно за анализу, јер је у тим ситуацијама оно ручно дефинисано.

Табела 3.1. служи за приказ основних информација везаних за коришћене алате са приказом сигурносних претњи за које имају подржану детекцију, које су наведене у табели 2.1.1. У наредним потпоглављима је сваки алат детаљније образложен, укључујући и начин њиховог покретања. Најчешће критичне сигурносне претње које су наведене у описима алата и нису укључене у оквиру тестова бенчмарка, као и претње укључене у оквиру тестова изузев SQLi, дефинисане су у прилогу В.

Табела 3.1. Основне информације везане за коришћене алате

Алат	Верзија алата	Подршка за сигурносне претње у тестовима бенчмарка										
		Инјекција команде	Слаб крипто. алгоритам	Слаба хеш функција	LDAP инјекција	Манипулација путањама	Колачић ...	SQL инјекција	Кршење граница ...	Слаб ген. псеудосл. вр.	XPATN инјекција	XSS
<i>Checkstyle</i>	2.17											
<i>FindBugs</i>	3.0.1					Да		Да				Да
<i>FindBugs</i> са прикључком <i>FindSecBugs</i>	1.4.6	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
<i>Lapse Plus</i>	2.8.1	Да			Да	Да		Да			Да	Да
<i>PMD</i>	5.5.1											
<i>VisualCode Grepper</i>	2.1.0		Да					Да		Да		Да
<i>Yasca</i>	2.2		Да	Да				Да		Да		Да

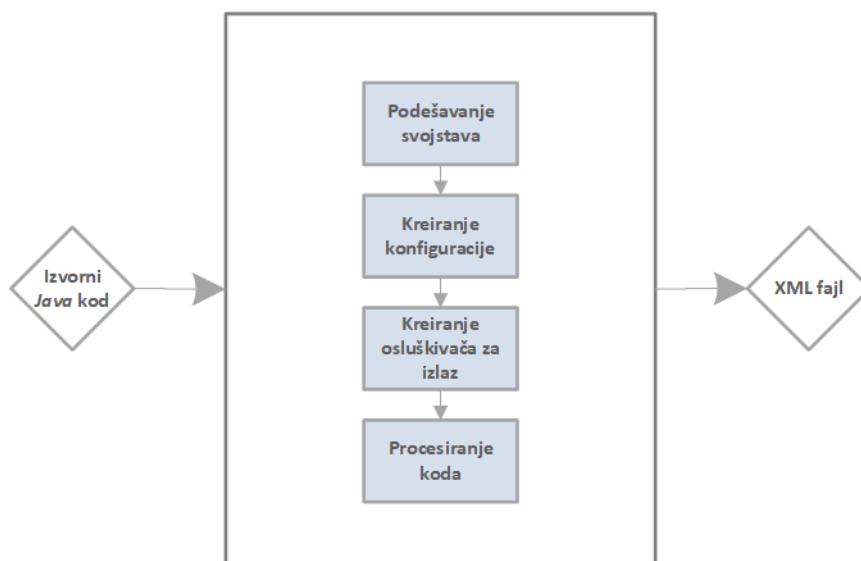
3.1. Checkstyle

Checkstyle представља алат за проверу изгледа изворног кода написаног у програмском језику *Java* према одговарајућем стандарду. Иако он не детектује грешке у коду, он представља значајан алат за побољшање писања кода према стандарду језика *Java*.

Овај алат пружа скуп конфигурисаних модула у оквиру стабла са кореним модулом *Checker*, где сваки модул има одређени број својстава који се односе на то како модули обављају провере. За свако својство модула се омогућава модификовање вредности својства у оквиру конфигурационог фајла уколико није задовољавајућа [13]. Поред тога, омогућено је додавање модула у виду провера (*FileSetChecks*), филтера (*Filters*) и ослушкивача (*AuditListeners*), где је провера модул који врши детекцију дефеката који се пријављују као догађаји. Како би се догађај проследио ослушкивачу, неопходно је да он прође скуп филтера који одлучују о даљем прослеђивању. У оквиру анализе фајлова, додати ослушкивачи прате статус догађаја, тако што добијају обавештења о детекцији проблема, уносу филтра или успешном проласку кроз филтер [14].

На слици 3.1.1 је приказан начин рада алата, који је дефинисан кодом алата тј. методом *runCheckstyle* у оквиру класе *Main*, која чини омотач алата. На основу пруженог изворног кода, врши се анализа истог која укључује следеће основне кораке: подешавање својстава (енгл. *Properties*), креирање конфигурације, креирање ослушкивача за излаз и процесирање кода. Први корак подразумева подешавања системских својстава, попут верзије окружења. Њему следи креирање конфигурације, која укључује својства из претходног корака и следеће атрибуте - број нити за корени модул *Checker* и индикатор којим се одређује да ли се извршавају сви модули. На крају се врши креирање ослушкивача, који је поменут у претходном параграфу, и процесирање кода. Овај последњи корак покреће корени модул *Checker* и подмодуле који се налазе у његовом стаблу, којим се врши анализа кода. Може се представити следећим корацима:

- 1) Обавештавање свих ослушкивача о започетом процесу;
- 2) Процесирање изворног кода користећи све модуле за провере *FileSetChecks*, који на основу дефинисаног кода у методи *processFiltered* врше назначене провере и
- 3) Обавештавање свих ослушкивача о завршеном процесу.



Слика 3.1.1. Дијаграм рада алата *Checkstyle*

Имплементирани модули у оквиру алата су груписани у оквиру четрнаест класа. Њих чине:

- анотације - везан за анотације над елементима језика попут недостатка кључних речи *Deprecated* и *Override*;
- провере блокова - односи се на изглед блокова и постојање окружујућих витичастих заграда;
- дизајн класа - односи се на изглед и својства класа (нпр. да треба да буде класа декларисана као *final* ако има само приватне конструкторе);
- кодирање - дефинише основна правила везана за изглед кода у програмском језику *Java* по стандарду попут присутности декларације пакета, изгледа методе *equals* за стрингове и нелегалних типова;
- заглавље - везан за изглед заглавља, где се може заглавље проверити у односу на задати регуларни израз;
- увози - дефинише изглед исказа за увоз, редувантност и контролу над увезеним садржајем укључујући да ли је дозвољен увоз;
- *Javadoc* коментари - односи се на изглед својстава алата *Javadoc* за генерисање документације у HTML формату из коментара наведених у коду;
- метрике - укључује дозвољену комплексност кода (нпр. цикломатска комплексност и комплексност према алату *JavaNCSS*) и дозвољени број инстанци других класа као поља класе и броја класа од којих зависи одређена класа;
- модификатори приступа - везан за редослед навођења и редувантност модификатора;
- конвенције за именовање - дефинише идентификаторе за одређене елементе кода у програмском језику *Java* попут константи, пакета и параметара метода;
- регуларни изрази - дефинише употребу одређеног регуларног израза која укључује пребројавање, простирање израза на једну или више линија кода и проверу назива фајлова;
- провере величина - односи се на различита пребројавања попут пребројавања линија кода фајла, дужине линије и метода
- празнине - односи се на неопходну и дозвољену употребу празнина у оквиру елемената кода односно токена и
- разно - укључује модуле који не припадају претходно наведеним класама попут изгледа дефиниције низа и индентације.

Покретање алата *Checkstyle* је омогућено коришћењем алата *Ant* и *Maven* у оквиру којих је интегрисан, али поред тога се може додати као прикључак у интегрисана развојна окружења попут *Eclipse* и *NetBeans*. У овом раду је покренут алат коришћењем *Maven* прикључка. Како би се генерисао извештај као део извештаја пројекта треба додатно дефинисати прикључак у оквиру „*pom.xml*“ конфигурационог фајла навођењем истог у оквиру дела за генерисање извештаја тј. елемента *reporting*. Изглед елемента за овај прикључак је приказан на слици 3.1.2.

Процес покретања алата извршава се скриптом која садржи команде приказане на слици 3.1.3. Овим командама се извршава *Checkstyle* анализа над целокупним кодом бенчмарка, где

се касније издвајају неопходни тест фајлови приликом парсирања генерисаног XML фајла. Поред тога, врши се бележење времена потребног за анализу.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.17</version>
  <reportSets>
    <reportSet>
      <reports>
        <report>checkstyle</report>
      </reports>
    </reportSet>
  </reportSets>
</plugin>
```

Слика 3.1.2. Елемент за *Checkstyle* извештаје у конфигурационом фајлу „*pom.xml*“

```
call mvn compile checkstyle:checkstyle -
Dbuildtime.output.csv=true -
Dbuildtime.output.csv.file=classes\out.csv
call mvn validate -Ptime -Dexec.args="checkstyle"
```

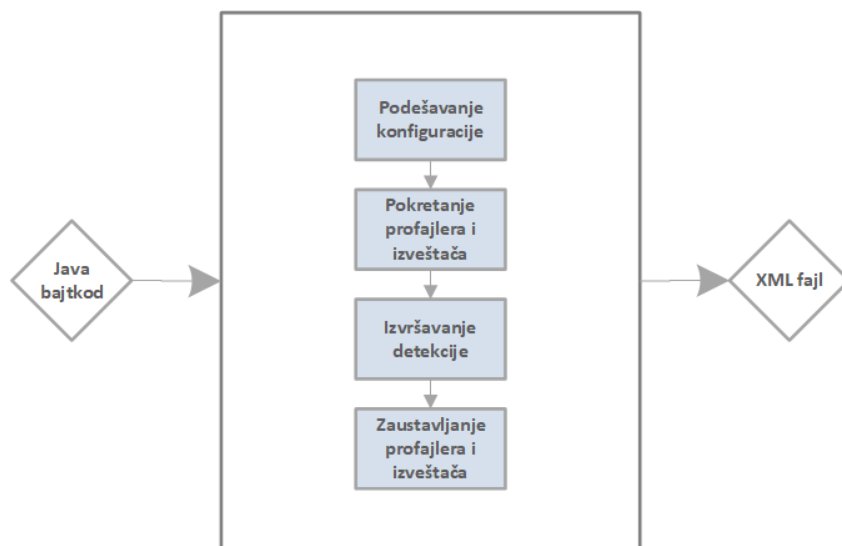
Слика 3.1.3. Команде за покретање алата *Checkstyle*

3.2. *FindBugs* и *FindSecBugs*

За разлику од алата *Checkstyle* поменутог у потпоглављу 3.2., алат *FindBugs* тражи грешке у *Java* бајткоду. Базиран је на методологији багова (енгл. *bug patterns*), који се појављују услед комплексних функционалности језика, неразумевања метода апликационог програмског интерфејса, неразумевања приликом измена кода у току одржавања и ситних грешака попут грешака при куцању [15]. На овај начин, анализа се обавља поклапањем узорака са бајткод инструкцијама, где се као резултат добијају не само грешке, већ и упозорења на потенцијалне проблеме попут оних везаних за перформансе.

На слици 3.2.1. је приказан дијаграм тока анализе овог алата, који наводи основне кораке у оквиру процеса детекције претњи. Они су издвојени из методе *FindBugs2.execute*, која је уочена као метода са најзначајнијим корацима. Почетним кораком, који се односи на подешавање конфигурације анализе, врше се инстанцирање профајлера, креирање компатибилног *BCEL* (енгл. *Byte Code Engineering Library*) слоја [15], подешавање режима за извештавање и креирање плана за извршавање (одређивање који детектори се укључују). Следећи корак обухвата покретање профајлера, односно покретање процеса мерења времена потребног за анализу, и покретање извештача, који акумулира све информације наведене приликом анализе (укључујући грешке). Њему следује комплекснија кључна фаза, извршавање детекције, која у оквиру једног или више пролаза врши анализу кода коришћењем одговарајућих детектора, где је број пролаза одређен бројем детектора и њиховим

предусловима. Први пролаз се односи на сакупљање информација о реферисаним класама, које се не извештавају од стране извештача. Међутим, уколико се врши више пролаза, онда први пролаз обухвата примену детектора над свим класама које реферишу апликационе класе, док се у наредним примењују детектори само над апликационим класама [15]. На крају се врши заустављање покренутог профайлера и извештача.



Слика 3.2.1. Дијаграм рада алата *FindBugs*

Свака инстанца грешке се сврстава у одговарајућу категорију. Њих чине следеће категорије грешака:

- лоша пракса за кодирање (нпр. поређење стрингова коришћењем оператора „=“);
- исправност кода (нпр. бесконачна петља);
- експериментална категорија (нпр. у оквиру методе није урађено чишћење ресурса или тока података);
- интернационализација (нпр. није коришћен *Locale* објекат у оквиру методе која користи специфичне карактере);
- претња услед злонамерног кода (нпр. метода *finalize* треба да има заштићен приступ, не јавни);
- исправност везана за вишенитно програмирање (нпр. нит је креирана без имплементације *run* методе);
- перформансе (нпр. објекат се пакује и одмах након тога распакује);
- сигурност (нпр. SQL инјекција) и
- непоуздан код (нпр. додела у оквиру наредбе *return* која нема ефекта).

Од наведених категорија, категорија сигурност је за овај рад од највећег значаја, где се издваја једанаест грешака укључујући рад са базом података, разбијање HTTP одговора на делове (енгл. *HTTP response splitting*), манипулација путањама (енгл. *Path Traversal*), SQL инјекцију и инјекцију клијентске скрипте односно XSS. Конкретно за SQL инјекцију постоје две грешке, где се прва односи на прослеђивање динамичког SQL упита *execute* или *addBatch* методама. Друга грешка подразумева генерисање пре-компајлиране наредбе користећи стринг који се може мењати, односно недостатак параметризованог дела упита.

Архитектура овог алата је заснована на прикључцима (енгл. *plugin*), што омогућава олакшано додавање произвољних детектора грешака у програмском језику *Java* [15]. Сваки детектор може да се користи за анализу више различитих багова.

Са друге стране, *FindSecBugs* односно *FindSecurityBugs* представља прикључак за *FindBugs* којем је циљ да побољша анализу кода са аспекта сигурности. Он омогућава детекцију 113 различитих типова претњи са преко 689 јединствених API (*Application programming interface*) потписа, где је за сваку претњу дефинисана референтна CWE вредност [16]. Примери претњи које детектује су: манипулација путањама, инјекција команде, *NULL* бајт инјекција, исцрпљеност ресурса, кршење граница поверења, LDAP инјекција и многе друге. За сигурносну претњу SQL инјекције коју такође детектује, овај алат је сврстава у оквиру више подтипова претњи SQL инјекције у зависности од пружених улазних параметара и коришћења апликационих програмских интерфејса попут *Turbine*, *Hibernate*, *JDBC*, *JPA* и *JDO*.

Покретање алата *FindBugs* се може учинити преко командне линије и коришћењем алата *Ant* и *Maven* у оквиру којих је интегрисан, али поред тога се може покренути у оквиру графичког корисничког интерфејса и интегрисаних развојних окружења попут *Eclipse* и *NetBeans* у виду прикључака. У овом раду је покренут алат коришћењем *Maven* прикључка. Како би се омогућио његов рад, неопходно га је конфигурисати у фајлу „*pom.xml*“, што је уграђено приликом инсталације бенчмарка. Прикључак *FindSecBugs* је такође додат дефинисањем одговарајућег профила.

Процес извршавања *FindBugs* анализе чини извршавање скрипте путем командне линије, која садржи команде приказане на слици 3.2.2. Овим командама се извршава *FindBugs* анализа над целокупним кодом бенчмарка, где се касније издвајају неопходни тест фајлови приликом парсирања генерисаног XML фајла. Поред тога, врши се бележење времена потребног за анализу. На исти начин се покреће *FindBugs* са прикључком *FindSecBugs* преко дефинисаног профила из конфигурационог фајла, што је приказано на слици 3.2.3.

```
call mvn compile findbugs:findbugs -
Dbuildtime.output.csv=true -
Dbuildtime.output.csv.file=classes\out.csv
call mvn validate -Ptime -Dexec.args="findbugs"
```

Слика 3.2.2. Команде за покретање алата *FindBugs*

```
call mvn compile -Pfindsecbugs -Dbuildtime.output.csv=true -
Dbuildtime.output.csv.file=classes\out.csv
call mvn validate -Ptime -Dexec.args="findsecbugs"
```

Слика 3.2.3. Команде за покретање алата *FindBugs* са прикључком *FindSecBugs*

3.3. *LAPSE Plus*

LAPSE Plus (*LAPSE+*) односно једноставна анализа сигурности програма (енгл. *Lightweight Analysis for Program Security*) представља алат за детекцију претњи у апликацијама написаним у програмском језику *Java*. То се омогућава коришћењем прикључка за

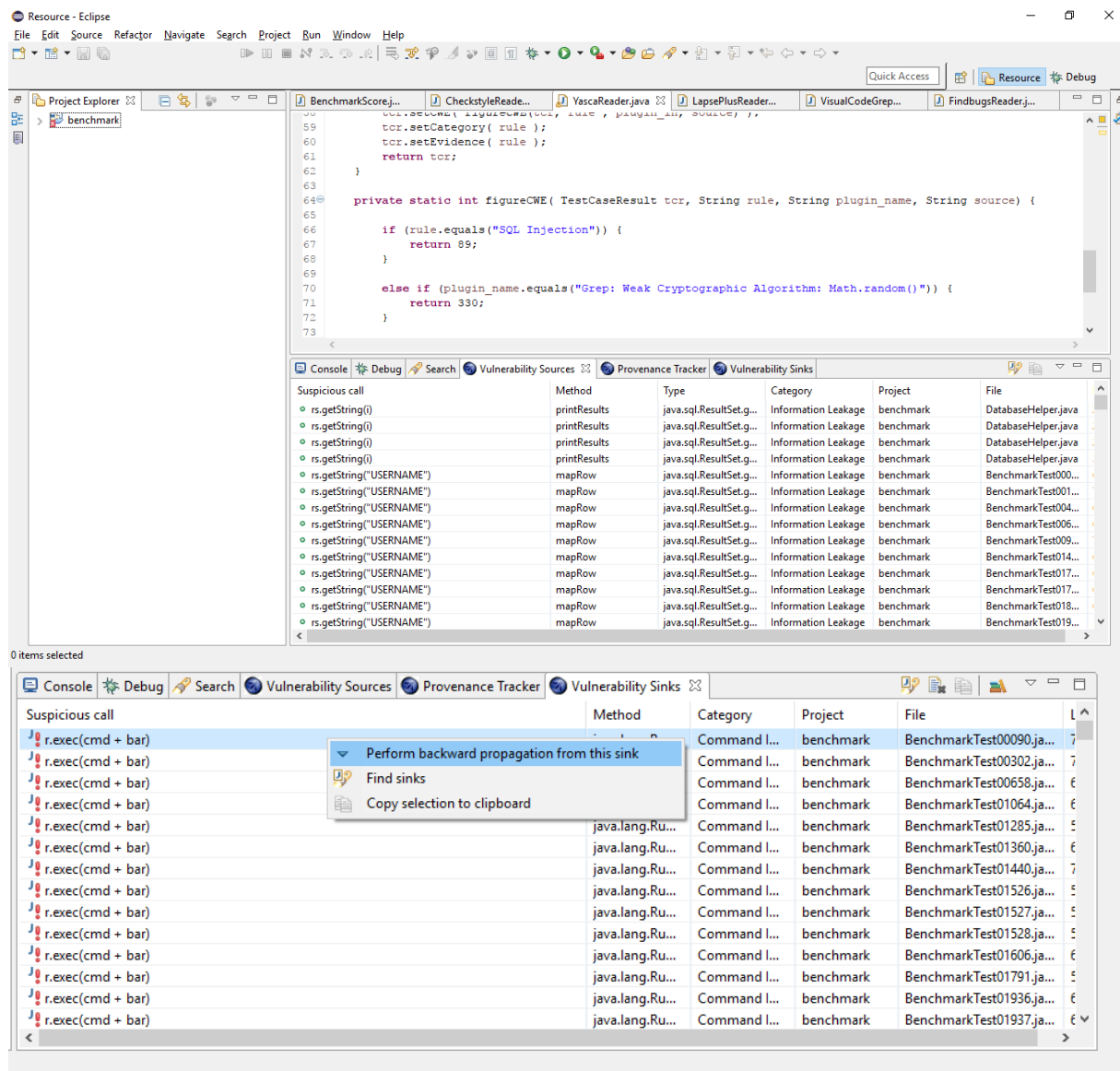
интегрисано развојно окружење *Eclipse*. Базиран је на статичкој анализи кода којим се врши проналажење извора и тачке извршавања претње. Извор се односи на део кода који представља извор напада, тј. у оквиру којег се врши инјекција неповерљивих података попут параметара у колачићу. Тачка извршавања претње односи на део кода који може бити мета инјекције података како би се манипулисало апликацијом (нпр. кроз одговор сервлета) [17]. Ако је могуће успоставити везу између извора и тачке извршавања претње, онда је сигурност апликације угрожена.

У складу са наведеним начином рада алата, може се закључити да он врши детекцију претњи у виду три корака - детекције извора претње (енгл. *Vulnerability Source*), детекције тачке извршавања претње (енгл. *Vulnerability Sink*) и проналажења порекла (енгл. *Provenance Tracker*). На слици 3.3.1. се може уочити преглед поменутих корака у оквиру окружења *Eclipse*, где се детектују претње у оквиру отвореног пројекта у окружењу. За прва два корака, приказују се детекције претње укључујући категорију, линију кода и линију у оквиру које се појављује претња, док се за пронажење порекла уочава стабло пропагације уназад у односу на одабрану тачку извршавања претње.

С обзиром да је у оквиру овог алата неопходно одабрати покретање одговарајућег корака у оквиру окружења *Eclipse*, за приказ процеса детекције је одабрана детекција тачака извршавања претње (енгл. *Vulnerability Sink*), чији је процес представљен дијаграмом на слици 3.3.2. Овај дијаграм је изведен на основу методе *ComputeSinksJob.run*, којом се покреће дефинисан посао. Њега чине следеће кључне фазе: учитавање пројекта у оквиру окружења за анализу, учитавање фајла „*sinks.xml*“ и провера кода пројекта у односу на дефинисане методе из претходно учитаног фајла. Фајл „*sinks.xml*“ наводи методе које нису безбедне за употребу и везује сваку за одређену категорију претње (нпр. *javax.persistence.EntityManager.createNativeQuery(String)* за SQL инјекцију). У оквиру главне фазе анализе се за сваки пројекат проверава постојање сваке од дефинисаних метода у претходно поменутом фајлу. За тај корак је кључно коришћење класе *org.eclipse.jdt.core.search.SearchEngine*, која пружа могућност враћања опсега за претрагу, који укључује директоријуме са изворним кодом, *JAR* (енгл. *Java Archive*) фајлове и референтне пројекте, и претраживања небезбедних метода по том опсегу.

Претње које *LAPSE+* детектује се односе на сигурносне претње, односно инјекцију непроверених и неповерљивих података. Свакој претњи се придружује одговарајућа категорија, којих има дванаест у оквиру овог алата. Њих чине:

- Промена параметара (енгл. *Parameter Tampering*);
- Промена веб-адресе (енгл. *URL Tampering*);
- Манипулација садржајем заглавља (енгл. *Header Manipulation*);
- Манипулација садржајем колачића (енгл. *Cookie Poisoning*);
- Инјекције које укључују оне везане за: SQL, команде, XPATH (*XML Path Language*) и LDAP;
- XSS - инјекција клијентске скрипте;
- Разбијање HTTP одговора на делове (енгл. *HTTP Response Splitting*) и
- Манипулација путањама [17].



Слика 3.3.1. Прегледи корака алата *LAPSE+* у оквиру окружења *Eclipse*

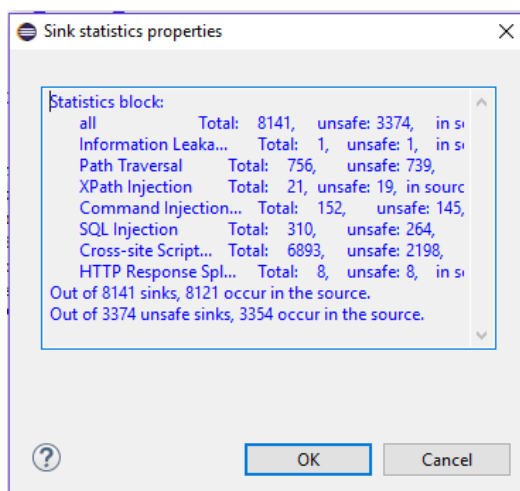


Слика 3.3.2. Дијаграм процеса детекције тачке извршавања претње алата *LAPSE+*

За претњу SQL инјекције се траже појаве функција које нису безбедне попут функција интерфејса `java.sql.Statement`, `java.sql.Connection` и `javax.persistence.EntityManager`. Потписи ових функција су наведени у фајлу „`sinks.xml`“, одакле се проверава поклапање са функцијама у коду теста [18].

Под изворима претњи из претходно наведене листе се сматрају промена параметара, манипулација садржајем заглавља, промена веб-адресе, манипулација садржајем колачића и додатно цурење података (енгл. *Information Leakage*), док већином остали чине тачке извршавања претње.

Како би се омогућило покретање овог алата, неопходно је инсталирати прикључак у оквиру окружења *Eclipse* према упутству наведеном за инсталацију прикључка [17]. Након тога ће бити омогућен приказ прегледа за претходно наведена три корака која обухвата овај алат. Поред ових прегледа, омогућен је избор опције за приказ статистике (енгл. *Get Sinks Statistics*), што је приказано на слици 3.3.3. Мана при покретању овог алата је недовољно меморије за целокупно анализирање пројекта кроз сва три корака величине коришћеног бенчмарка, односно искључен је корак пропагације уназад услед грешке недостатка меморије. Због тога, преузета је листа потенцијалних претњи из прегледа *Vulnerability Sink*.



Слика 3.3.3. Приказ статистике алата *LAPSE+*

Како би се пружио XML фајл бенчмарку од стране овог алата, неопходно је самостално извршити конверзију текстуалног фајла у XML фајл, што је постигнуто имплементацијом класе *ToXML* у програмском језику *Java*. На основу улазног текстуалног фајла, ова класа умеће одговарајућу индентацију и атрибуте на тачно одређене позиције попут извора, назива фајла, броја теста, категорије и CWE вредности. Код овог конвертора се налази у прилогу рада у оквиру кода класе *ToXML* (Прилог А.1).

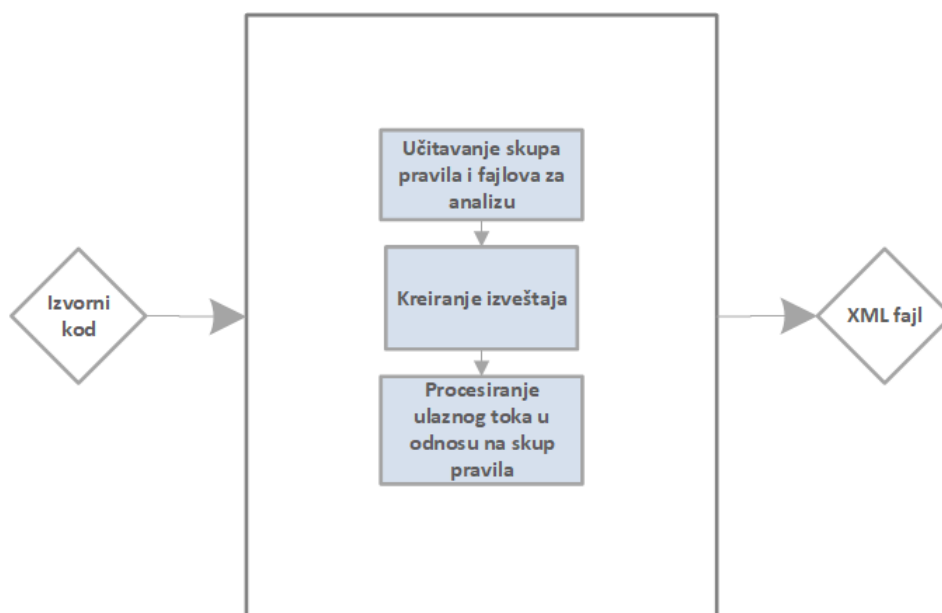
3.4. *PMD*

Алат *PMD* врши анализу кода тиме што проверава синтаксу изворног кода у односу на дефинисана правила. За разлику од претходно наведених алата, он подржава следеће алате и програмске језике: *Java*, *JavaScript*, *Apex*, *Visualforce*, *PLSQL*, *Apache Velocity*, *XML* и *XSL*. Он додатно укључује и детектор *CPD* (енгл. *copy-paste-detector*), који проналази дубликате у коду у програмским језицима попут језика *Java*, *C*, *C++*, *C#*, *PHP*, *Python* и других [19]. На овај начин, појава исте грешке на више места се лако уклања. Анализа коју он пружа је заснована

на скупу правила којим се дефинише начин исправног писања кода, али не садржи детекцију сигурносних претњи, што га не чини мање ефикасним.

На слици 3.4.1. је приказан дијаграм са основним корацима процеса детекције претњи коришћењем овог алата. Први корак прослеђивања фајла и скупа правила (енгл. *RuleSet*) у оквиру алата је део постављања конфигурације, која укључује и одлучивање у ком програмском језику су написани фајлови на основу њихових екстензија. Овом кораку следи креирање извештаја на основу прослеђеног скупа правила. Последња фаза овог процеса укључује више комплексних корака, који су битни за вршење анализе. Њих чине:

- 1) Прослеђивање улазног тока (енгл. *InputStream*) везаног за фајл генератору парсера JavaCC (*Java Compiler Compiler*);
- 2) Враћање резултата алату од стране JavaCC у виду референце на апстрактно синтаксно стабло;
- 3) Прослеђивање апстрактног синтаксног стабла од стране алата слоју табеле симбола, која креира опсеге, проналази декларације и употребе;
- 4) Прослеђивање апстрактног синтаксног стабла од стране алата слоју за анализу тока података, који креира графове управљачког тока и чворове тока података (у случају да неко правило захтева анализу тока података) и
- 5) Обилазак апстрактног синтаксног стабла од стране сваког правила у оквиру скупа правила и вршење адекватне провере, где може правило да приступи табели симбола и чворовима тока података [19].



Слика 3.4.1. Дијаграм рада алата *PMD*

Скупови правила за програмски језик *Java* су везани за опште грешке при писању кода (нпр. креирање инстанци класе *Boolean*), комплексност, тестирање, рад са стринговима, оптимизацију и друге аспекте писања кода. Поред тога, могу се имплементирати и уградити додатни скупови правила коришћењем технологија *Java* и ХРАТН. Међутим, писање правила у програмском језику *Java* је једноставнији процес, али и бржи с обзиром на прекомпилацију правила [14].

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
  <version>3.6</version>
  <configuration>
    <linkXref>true</linkXref>
    <targetJdk>1.7</targetJdk>
  </configuration>
</plugin>

```

Слика 3.4.2. Елемент за *PMD* извештаје у фајлу „*pom.xml*“

Покретање алата *PMD* је омогућено коришћењем прикључака у оквиру окружења и алата *Maven*, *Gradle*, *Eclipse*, *NetBeans*, *JBuilder*, *JDeveloper* и *IntelliJ IDEA*. Поред тога, овај алат се може покренути преко командне линије и помоћу алата *Apache Ant* [19]. У овом раду је покренут алат коришћењем *Maven* прикључка. Како би се генерисао извештај као део извештаја пројекта треба додатно дефинисати прикључак у оквиру „*pom.xml*“ конфигурационог фајла навођењем истог у оквиру дела за генерисање извештаја тј. елемента *reporting*. Изглед елемента за овај прикључак је приказан на слици 3.4.2.

Покретање самог алата се извршава скриптом која садржи команде приказане на слици 3.4.3. Овим командама се извршава *PMD* анализа над целокупним кодом бенчмарка, где се касније издвајају неопходни тест фајлови приликом парсирања генерисаног XML фајла. Поред тога, врши се бележење времена потребног за анализу, као и код претходно наведених уграђених алата у оквиру бенчмарка.

```

call mvn compile pmd:pmd -Dbuildtime.output.csv=true -
Dbuildtime.output.csv.file=classes\out.csv
call mvn validate -Ptime -Dexec.args="pmd"

```

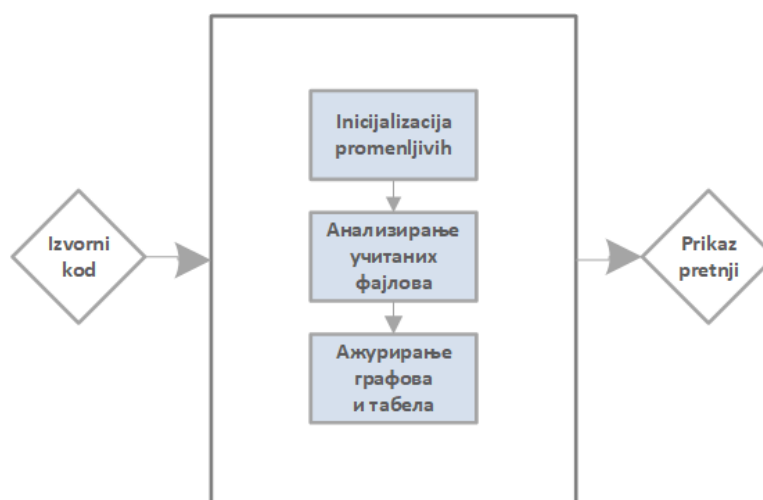
Слика 3.4.3. Команде за покретање алата *PMD*

3.5. *VisualCodeGrepper*

Алат *VisualCodeGrepper* је алат за статичку анализу кода који има намену да пронађе сигурносне претње. Користи се за код написан у следећим програмским језицима: *C++*, *C#*, *Visual Basic*, *PHP*, *Java* и *PL/SQL*. За сваки поменути језик, он користи посебан конфигурациони фајл у којем се наводе небезбедне функције и пакети, при чему може да модификује његов садржај. За програмски језик *Java*, користи се конфигурациони фајл „*javafunctions.conf*“, где се за сваку функцију или пакет могу дефинисати ниво озбиљности, који је накнадно образложен, и опис претње. Поред тога, он пружа могућност додавања привременог регуларног израза који ће се поредити са садржајем кода у току тренутне сесије апликације.

Процес анализе кода овог алата представљен је дијаграмом на слици 3.5.1, где су наведене кључне фазе. Први корак чини иницијализација променљивих, односно објекта

rtResultsTracker, који је инстанца класе *ResultsTracker*. У оквиру њега се бележе резултати анализе, број фајлова за анализу и сами фајлови, као и бројачи везани за индивидуалне фајлове и директоријум попут броја линија кода, коментара, *white space* карактера и небезбедних функција. Друга фаза, односно анализа учитаних фајлова, врши се тако што се свака линија фајла засебно анализира. При том поступку се ажурирају бројачи везани за број анализираних линија кода (за цео директоријум и за засебан фајл), али и бројачи који су везани за број коментара (уколико учитана линија садржи коментаре) и *white space* карактере (уколико учитана линија садржи *white space* карактере). Након тога се фајлови анализирају у односу на провере везане за програмски језик у којем су написани. За програмски језик *Java* се користи фајл „*modJavaCheck.vb*“, који проверава линију кода у односу на дефинисане претње које су у наставку образложене. Последњи корак чини ажурирање графова и табела у оквиру графичког корисничког интерфејса приказаних на слици 3.5.3., где се памте забележени бројачи и резултати.



Слика 3.5.1. Дијаграм рада алата *VisualCodeGrepper*

Свака претња категоризована је према класи сигурносне претње којом је обухваћена. На основу садржаја фајла „*modJavaCheck.vb*“ [20] може да се закључи да овај алат детектује следеће сигурносне претње:

- Употреба функције *Thread.sleep()* у оквиру сервлета укључујући случај утркивања;
- SQLi;
- XSS;
- Лоша валидација улазних података;
- Лоша употреба функције *java.lang.Runtime.exec()*;
- Лош HTTP захтев;
- Небезбедна имплементација клонирања;
- Небезбедна имплементација серијализације;
- Употреба јавних променљивих у класама;
- Лоша расподела ресурса нитима;
- Потенцијални приступ коду са највећим привилегијама;
- Лоши називи привремених фајлова;

- Лоша употреба *RequestDispatcher* објекта (у комбинацији са корисничком променљивом);
- Небезбедна XML експанзија ентитета;
- Лоша употреба примитивних типова и
- Ресурси нису безбедно обрађени у оквиру блокова за обраду изузетака.

Овај алат наводи детекцију претњи везаних за SQL инјекцију у случајевима приказаним на слици 3.5.2. Може се уочити да он дефинише као примарни знак претње употребу променљиве која није обрађена (енгл. *unsanitised*), где се касније садржај поруке о претњи мења у зависности од места дефинисања упита. Додатно је уочено да овај алат неће извршити анализу уколико препозна да је у апликацији коришћен валидатор попут фрејмворка за валидацију *Apache Struts* [20].

Поред тога, свака претња одређена је врстом озбиљности претње (енгл. *Severity*). Према редоследу од највеће до најмање озбиљности, нивое озбиљности чине: критични (енгл. *Critical*), високи (енгл. *High*), средњи (енгл. *Medium*), стандардни/нормални (енгл. *Standard/Normal*), ниски (енгл. *Low*), потенцијална претња/најбоља пракса (енгл. *Potential Issue/Best Practice*) и сумњив коментар који је индикатор лошег кода (енгл. *Suspicious comment indicating broken code*).

За разлику од претходних алата, овај алат има искључиво опцију покретања анализе у оквиру графичког корисничког интерфејса самог алата. Након инсталације и покретања програма, неопходно је дефинисати програмски језик у којем је написан код који се анализира. Ово се постиже избором опције „Подешавања“ (енгл. *Settings*) у главном менију у прозору, која приказује доступне језике. Додавање директоријума у којем се налазе фајлови за тестирање се врши избором опције „Фајл“ (енгл. *File*), а затим у зависности од тога да ли је у питању фајл или директоријум се бира опција „Нова мета - директоријум“ (енгл. *New target directory*) или „Нова мета - фајл“ (енгл. *New target file*). Претходно наведени кораци резултују учитавањем фајлова за анализирање, што омогућава покретање анализе избором опција „Претрага“ (енгл. *Scan*) и „Детаљна претрага“ (енгл. *Full scan*) према наведеном редоследу. Поред опције детаљне претраге могуће је извршити следеће врсте претрага [20]:

- Претрага искључиво коментара (енгл. *Comments Only*) - коментари могу да укажу на недовршени или лош код, стога се претражују према шеснаест фраза попут „*ToDo*“ и „*FixMe*“;
- Претрага искључиво кода (енгл. *Code Only*) - претрага потенцијалних сигурносних претњи и опасних функција наведених у конфигурационом фајлу које се могу наћи у коду (може да се модификује) и
- Претрага искључиво опасних функција (енгл. *Dangerous Functions Only*) - претрага функција наведених у конфигурационом фајлу које се могу наћи у коду (може да се модификује).

Након извршене анализе, приказује се прозор „Преглед кода“ (енгл. *Code breakdown*) видљив на слици 3.5.3., који графичким приказом истиче број линија анализираних кода и празнина, број коментара, број потенцијално недовршених делова и број потенцијалних претњи.

Поред прегледа приказаног на слици 3.5.3, резултати анализе се могу уочити у оквиру дела за резултате (енгл. *Results*) и дела за сумарну табелу (енгл. *Summary Table*). Оба прегледа излажу поред линије и фајла где је претња пронађена и дефиницију претње, која укључује

назив категорије претње и озбиљност претње, с тим да је у оквиру сумарне табеле резултат претраге приказан скраћено и табеларно, што се може уочити на слици 3.5.4. која приказује део за резултате. Такође је значајно што се коришћењем табеле могу сортирати колоне попут фајла, приоритета и врсте претње. Добијени резултати се на крају могу сачувати у виду текстуалног, XML или CSV (*Comma-separated values*) преко наведених опција под ставком „Фајл“ (енгл. *File*). За бенчмарк који се проучава у овом раду, прослеђује се генерисани XML фајл.

```
'== Is unsanitised dynamic SQL statement prepared beforehand? ==
If CodeLine.Contains("=") And (CodeLine.ToLower.Contains("sql") Or CodeLine.ToLower.Contains("query")) And (CodeLine.Contains("'''") And CodeLine.Contains("+")) Then
    '== Extract variable name from assignment statement ==
    strVarName = GetVarName(CodeLine)
    ctCodeTracker.HasVulnSQLString = True
    If Not ctCodeTracker.SQLStatements.Contains(strVarName) And _
        (Not (strVarName.Contains("(") Or strVarName.Contains(")") Or strVarName.Contains(":") Or strVarName.Contains(";") Or strVarName.Contains(",") Or strVarName.C
End If

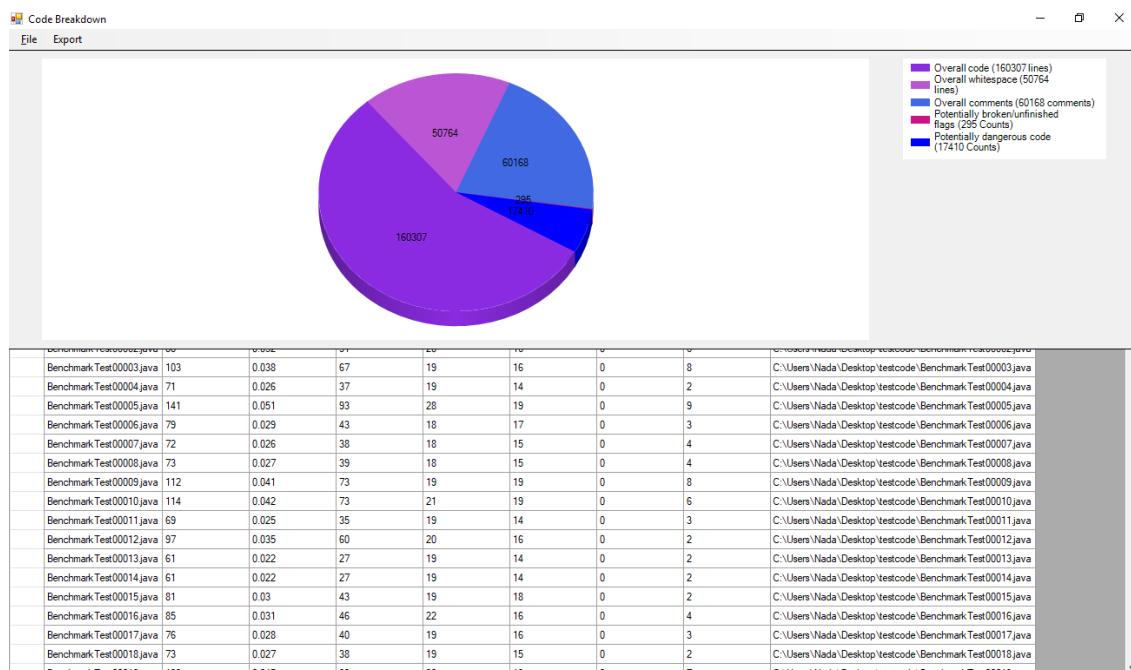
If Regex.IsMatch(CodeLine, "validate|encode|sanitize|sanitise") Then

    '== Remove any variables which have been sanitised from the list of vulnerable variables ==
    If ctCodeTracker.SQLStatements.Count > 0 Then
        For Each strVar In ctCodeTracker.SQLStatements

            If Regex.IsMatch(CodeLine, strVar & "\s*\-\s*\S*validate|encode|sanitize|sanitise\S*" & strVar) Then
                ctCodeTracker.SQLStatements.Remove(strVar)
                Exit For
            End If
        Next
    End If
ElseIf Regex.IsMatch(CodeLine, "\S*\.(prepareStatement|executeQuery|query|queryForObject|queryForList|queryForInt|queryForMap|update|getQueryString|executeQuery|" +
    "createNativeQuery|createQuery)\s*" & "(") Then

    '== Check usage of java.sql.Statement.executeQuery, etc. ==
    If CodeLine.Contains("'''") And CodeLine.Contains("+") Then
        '== Dynamic SQL built into connection/update ==
        frmMain.ListCodeIssue("Potential SQL Injection", "The application appears to allow SQL injection via dynamic SQL statements. No validator plug-ins were locate
    ElseIf ctCodeTracker.HasVulnSQLString = True Then
        '== Otherwise check for use of pre-prepared statements ==
        For Each strVar In ctCodeTracker.SQLStatements
            If CodeLine.Contains(strVar) Then
                frmMain.ListCodeIssue("Potential SQL Injection", "The application appears to allow SQL injection via a pre-prepared dynamic SQL statement. No validate
            End If
        Next
    End If
Next
```

Слика 3.5.2. Део кода алата *VisualCodeGrepper* за детекцију SQL инјекције



Слика 3.5.3. Преглед анализираног кода за алат *VisualCodeGrepper*

Priority	Severity	Title	Description	FileName	Line
7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is consid...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	30
2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	49
4	Standard	FileInputStream	This function acts as an entry point for external data and the code should be ma...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	63
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	63
4	Standard	FileInputStream	This function acts as an entry point for external data and the code should be ma...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	67
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	67
5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	72
4	Standard	FileInputStream	This function acts as an entry point for external data and the code should be ma...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	75
4	Standard	FileInputStream	This function acts as an entry point for external data and the code should be ma...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00001.java	77
7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is consid...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00002.java	30
2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00002.java	49
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00002.java	63
4	Standard	java.io.FileOutputStream	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00002.java	63
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00002.java	68
4	Standard	java.io.FileOutputStream	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00002.java	68
7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is consid...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	30
2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	49
4	Standard	getResourceAsStream	This function acts as an entry point for external data and the code should be ma...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	64
5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	73
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	84
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	85
4	Standard	java.io.PrintWriter	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	86
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00003.java	86
7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is consid...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00004.java	30
2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00004.java	49
7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is consid...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	30
2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	46
2	High	Poor Input Validation	The application appears to use data contained in the HttpServletRequest witho...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	47
3	Medium	Cipher.getInstance("DES	The code appears to use the DES algorithm. This is no longer considered secur...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	65
5	Low	Operation on Primitive Data Type	The code appears to be carrying out a mathematical operation on a primitive dat...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	79
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	89
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	90
4	Standard	java.io.PrintWriter	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	91
4	Standard	java.io.File	This functionality acts as an entry point for external data and the code should be...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00005.java	91
7	Potential Is...	Public Class Not Declared as Final	The class is not declared as final as per OWASP recommendations. It is consid...	C:\Users\Nada\Desktop\testcode\BenchmarkTest00006.java	30

Слика 3.5.4. Преглед резултата анализе за алат *VisualCodeGrepper*

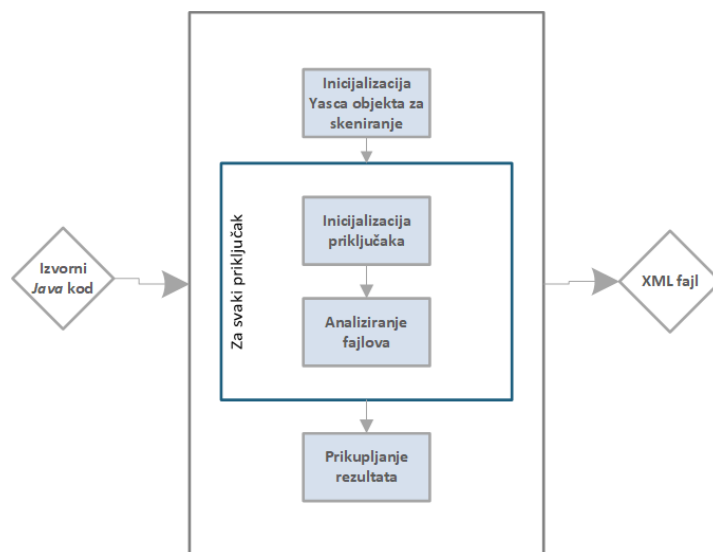
3.6. *Yasca*

Yasca, што је скраћеница за „још један анализатор изворног кода“ (енгл. *Yet Another Source Code Analyzer*), представља алат који детектује сигурносне претње, мери квалитет кода и перформансе и анализира начин писања кода. У односу на све поменуте алате у овом поглављу, једино он поред својих детектора комбинује екстерне алате. На овај начин се пружа смањена дупликација постојећих правила, додатна сигурност и подршка различитим језицима, јер су ти алати познати за извршавање статичке анализе кода. Могу се интегрисати алати разматрани у претходним поглављима *FindBugs* и *PMD*, али такође и *JLint*, *JavaScript Lint*, *PHPLint*, *Cppcheck*, *ClamAV*, *Pixy* и *RATS*. На основу тога се може закључити да *Yasca* пружа подршку за следеће програмске језике: *.NET*, *ASP*, *C/C++*, *COBOL*, *ColdFusion*, *CSS*, *HTML*, *Java*, *JavaScript*, *Perl*, *PHP*, *Python* и *Visual Basic*, али поред тога може да се користи и за скенирање HTTP саобраћаја.

Архитектура овог алата је базирана на прикључцима, где се разликују већи и мањи прикључци (енгл. *Major and Minor Plug-in*). Већи прикључци захтевају извршавање анализе од стране екстерних алата, док мањи користе уграђену логику која је надоградива. Како би се лако имплементирала и уградила нова правила, користи се посебан прикључак *Grep*. Захваљујући њему, додавање нових правила захтева само дефинисање регуларног израза, типа фајлова на који се примењују и назива правила и смештање истог у оквиру директоријума прикључци (енгл. *plugins*).

На слици 3.6.1. је приказан дијаграм рада овог алата, који се односи на основне кораке процеса детекције. Прву фазу чини иницијализација *Yasca* објекта за скенирање, који обухвата следеће кораке: проверу валидности фајлова за анализирање (да ли постоје и да ли могу да се читају) и проверу постојања прикључака. Након тога се за сваки прикључак врше кораци иницијализације и анализирања фајлова. Фазу иницијализације прикључака чине прослеђивање фајлова за обраду и покретање анализе над прослеђеним фајловима. Након

завршетка анализе од стране прикључака, бележе се резултати који су прикључци проследили *Yasca* објекту.



Слика 3.6.1. Дијаграм рада алата *Yasca*

Од свих екстерних алата који се могу уградити је најбољи за пролажење сигурносних претњи алат *FindBugs*, за којег је детаљније у потпоглављу 3.2. образложено које категорије претњи покрива. Поред ових претњи, алат *Yasca* покрива следеће претње у оквиру своје архитектуре:

- Коришћење технологије AJAX (*Asynchronous JavaScript And XML*), што може утицати на сигурност и перформансе);
- Фиксно кодирани подаци везани за базу података и е-мејл адресу;
- Детекција задњих врата (злонамеран софтвер);
- Логовање осетљивих информација;
- XSS;
- SQL инјекција (укључујући приказ упита као осетљивог садржаја);
- Форматирање кода попут исправног датума и исправног поређења стрингова;
- Референца на локални ресурс (путања);
- DoS (*Denial of service*);
- Коришћење слабог криптографског алгоритма (укључујући слаб генератор псеудослучајних вредности и фиксно кодиран криптографски кључ);
- Коришћење стандардног излазног тока уместо логовања;
- Коришћење опасних функција попут *Servlet.sleep()*, *Class.forName()*, *Runtime.exec()* и *System.loadLibrary()*;
- Коришћење података за које није извршена валидација у оквиру сесије;
- Коришћење лозинке која је једноставна или смештена без енкрипције;
- Коришћење параметра за дебаговање или скривеног поља форме;
- Недостатак провере на вредност *null* и
- Појава грешке за један у *For* петљи.

Претходно наведене претње су изложене у оквиру фајлова везаних за прикључак *Grep*, односно оне су једноставно додате у оквиру алата у виду одговарајућих регуларних израза. За случај SQL инјекције, регуларни изрази у фајлу „*Injection.SQL.grep*“ су приказани на слици 3.6.2., који детектују претњу засновану на пруженим необрађеним улазним подацима од стране корисника.

```
;Java Vulnerabilities
grep = /\\"(select|delete) .*from .*\+/i
grep = /\\"insert\s+into\.* .*\+/i
grep = /\\"update.*set.*\+/i
grep = /\\".*call .*\\"\\s*\+\\s*(req(uest)?)\.getParameter/i
grep = /prepareCall.*\\".*call .*\\"\\s*\+\\s*[a-zA-Z0-9_]\+/i

grep = /\\"(select|delete) .*from .*'\$_(GET|POST|REQUEST)\[/i
grep = /\\"(select|delete) .*from .*'\.\\s*\$_(GET|POST|REQUEST)\[/i
grep = /\\"select ([a-z0-9_]*))\\s*\$_(GET|POST|REQUEST)\[/i
grep = /\\"select ([a-z0-9_]*))\\s*\\"\\s*\.\\s*\$_(GET|POST|REQUEST)\[/i
```

Слика 3.6.2. Регуларни изрази за SQLi алата *Yasca*

Инсталацијом директоријума који садржи извршни фајл алта нису уграђени екстерни прикључци, стога је потребно и њих инсталирати и сместити у директоријум којем ће алат имати приступ. Покретање овог алата врши се одговарајућим командама путем командне линије или преко *Windows* графичког корисничког интерфејса. Међутим, препоручује се коришћење путем командне линије, јер пружа више опција и боље перформансе [21]. На слици 3.6.3. приказане су опције које су пружене кориснику приликом покретања. Овај алат пружа могућност одабира покретања искључиво дефинисаних прикључака и врсте генерисаног извештаја, који може бити у форматима HTML, CSV, XML и *SQLite*.

```
PS C:\yasca> .\yasca
Yasca-Core version 2.2
Copyright (c) 2010 Michael V. Scovetta. See docs/LICENSE for license information.

Usage: yasca [options] directory
Perform analysis of program source code.

--debug                additional debugging
-d "QUERYSTRING"       pass the query string to Yasca's sub-components
-h, --help             show this help
-i, --ignore-ext EXT,EXT ignore these file extensions
                        (default: exe,zip,jpg,gif,png,pdf,class)
--ignore-file FILE     ignore findings from the specified xml file
--source-required      only show findings that have source code available
-f, --fixes FILE       include fixes, written to FILE (default: not included)
                        (EXPERIMENTAL)
-l, --level LEVEL      show findings at least LEVEL (1-5) (default: 5=all)
-o, --output FILE      write output to FILE (default: unique file on
                        desktop in Yasca directory)
-p, --plugin DIR|FILE  load plugins from DIR or just load FILE (default: ./plugins)
-px PATTERN[,PATTERN...] exclude plugins matching PATTERN or any of "PATTERN,PATTERN"
                        (multiple patterns must be enclosed in quotes)
--log FILE             write log entries to FILE
-sa,--sa_home DIR      use this directory for 3rd party plugins (default: $SA_HOME)
-r, --report REPORT    use REPORT template (default: HTMLGroupReport). Options
                        include HTMLGroupReport, CSVReport, XMLReport, SQLReport,
                        DetailedReport, and ConsoleReport.
-s, --silent           do not show any output
-v, --version          show version information

Examples:
yasca c:\source_code
yasca /opt/dev/source_code
yasca -px FindBugs,PMD,Antic,JLint /opt/dev/source_code
yasca -o c:\output.csv --report CSVReport "c:\foo bar\quux"
yasca -d "SQLReport.database=./my.db" -r SQLReport /opt/dev/source_code
```

Слика 3.6.3. Преглед опција приликом покретања алата *Yasca*

4. АНАЛИЗА ТЕСТОВА

За анализу и међусобно поређење алата за детекцију претњи, неопходно је да постоји адекватан скуп тестова који ће моћи да издвоји ефикасније алате, као и да пружи одређене статистичке податке на основу тог скупа. Стога се може закључити да је од највеће важности да тестови буду различити према начину прослеђивања и садржају улазних података, као и да покривају што већи опсег начина напада у оквиру дефинисане претње.

Скуп тестова уграђених у код бенчмарка намењени су за детекцију сигурносних претњи наведених у табели 2.1.1. Примарна намена ових тестова је утврђивање успешности детекције правих сигурносних претњи које могу угрозити апликацију написану у програмском језику *Java*. Њихова успешност се одређује на основу следећа четири могућа исхода:

- 1) Стварно позитивни тј. TP (*True Positive*) - алат успешно детектује праву сигурносну претњу;
- 2) Лажно негативни тј. FN (*False Negative*) - алат није детектовао праву сигурносну претњу;
- 3) Стварно негативни тј. TN (*True Negative*) - алат успешно занемарује лажну сигурносну претњу и
- 4) Лажно позитивни тј. FP (*False Positive*) - алат није занемарио лажну сигурносну претњу.

Ови исходи израчунавају вредност резултата прецизности у оквиру бенчмарка (енгл. *Benchmark Accuracy Score*) приказану у оквиру једначине 4.3., која се налази у опсегу од нула до сто. Ова вредност представља Јуданов индекс (енгл. *Youden Index*), односно „Ј индекс“, који служи за мерење прецизности перформанси [22]. Израчунава се тако што се од збира параметара сензитивности и специфичности теста одузме број један, како би се нормализовала вредност. Појам сензитивности теста (дефинисан у једначини 4.1). односи се на стварно позитивну стопу TPR (*True Positive Rate*), која представља однос позитивних исхода који су идентификовани као позитивни, док се појам специфичности теста (дефинисан у једначини 4.2.) израчунава као разлика броја један и стварно негативне стопе FPR (*False Positive Rate*), која представља однос негативних исхода који су идентификовани као негативни.

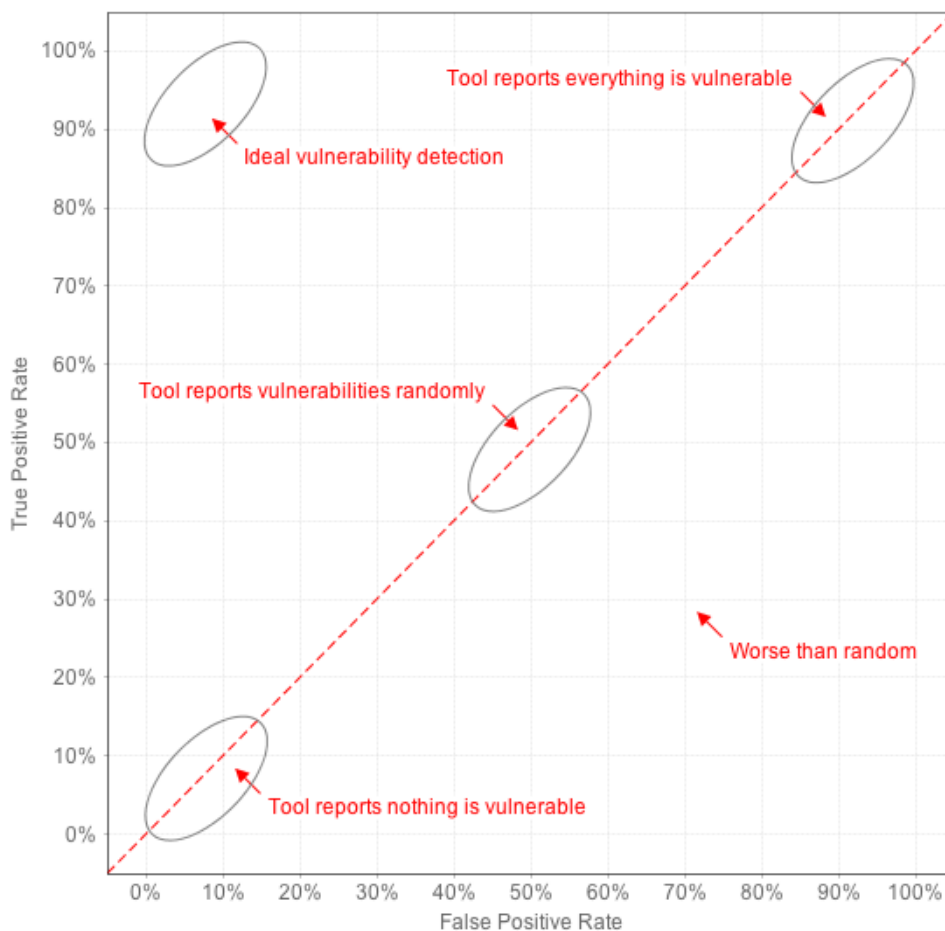
$$TPR = \frac{TP}{TP + FN} \rightarrow \text{Senzitivnost} = TPR \quad (4.1.)$$

$$FPR = \frac{FP}{FP + TN} \rightarrow \text{Specifičnost} = 1 - FPR \quad (4.2.)$$

$$\text{Benchmark Accuracy Score} = \text{Senzitivnost} + \text{Specifičnost} - 1 \quad (4.3.)$$

Вредност резултата прецизности за сваки алат се графички представља у бенчмарку у виду дијаграма, који приказује како је алат прошао анализу дефинисану скупом тестова. На слици 4.2. [3] је приказан поменути визуелни индикатор, на чијим се осама могу уочити стварно позитивна и лажно позитивна стопа. Сваки алат са резултатом се представља тачком, чија удаљеност од дијагонале представља вредност резултата прецизности. Уколико је лажно

позитивна стопа већа од стварно позитивне стопа, вредност резултата прецизности ће бити негативна. Најбољи резултат имаће алат чија се тачка налази у горњем левом углу графика.



Слика 4.2. Визуелни индикатор успешности алата [3]

Са становишта имплементације, тестови представљају класе у програмском језику *Java*, односно сервлете, који се односе на издвојене делове кода из апликација са претњама који би се појавили у реалној ситуацији. Тиме је покретање тестова бенчмарка једноставније и брже, што не би био случај да су коришћене целокупне апликације за тестирање. Поред фајлова са кодом тестова написаних у програмском језику *Java*, придружују им се XML фајлови, који садрже следеће метаподатке: верзију бенчмарка, категорију на коју се тест односи, број теста, индикатор праве претње и CWE вредност. Како би се приликом тестирања фајлова са тестовима утврдило да ли је алат детектовао праву претњу, не користи се поменути фајл са метаподацима, јер би његово коришћење успорило процес извршавања анализе. Уместо тога се користи додатни CSV фајл „*expectedresults-1.2.csv*“, где се број 1.2 односи на верзију коришћеног бенчмарка. Овај фајл чува за све тестове бенчмарка информације попут назива теста, категорије, индикатора праве претње и CWE вредност, чиме је приликом провере алата неопходно да се учитава овај фајл линију по линију.

С обзиром да је у овом раду нагласак на сигурносну претњу SQLi, детаљније су изучени тестови везани за њу. Из табеле 2.1.1. може да се закључи да је ова претња најраспрострањенија, јер је за њу везан највећи број тестова у односу на остале претње. Самим тим, изучавање ове претње је комплексније, јер постоји више начина за манипулисање улазним подацима. У поглављу 4.1. су описани постојећи тестови бенчмарка за SQL инјекцију,

односно на који начин они покривају наведену претњу. Поред тога је наведено који случајеви нису покривени и на који начин их је било могуће уградити у код бенчмарка у поглављу 4.2. У поглављу пет су истакнути резултати алата који су уочени након покретања описаних тестова у овом поглављу.

4.1. Тестови бенчмарка за SQL инјекцију

Тестови за SQL инјекцију треба да покрију два главна случаја у којима се она испољава. Њих чине:

- 1) Улазни подаци које апликација треба да обради су добијени од извора чији идентитет није проверен и
- 2) Пружени улазни подаци се користе на такав начин да се помоћу њих динамички конструише SQL упит [2].

Приликом анализе тестова везаних за ову претњу који се налазе у оквиру бенчмарка, издвојила су се следећа два различита атрибута која су везана за обраду података. Они су дефинисани врстом упита која се извршава и извором инјекције.

Под врстом упита се мисли на рад који се врши над табелом базе података, односно да ли је у питању уметање, селекција, ажурирање, брисање података, позив процедуралне функције или чак брисање целе табеле. Над скупом постојећих тестова су извршени искључиво упити везани за уметање и селекцију података и позив процедуралне функције, где су занемарени ажурирање и брисање података, што је сматрано неопходном надоградњом примењеном у оквиру овог рада.

Други атрибут односно извор инјекције представља начин достављања улазних података који се односи на део апликације коришћен од стране корисника као улазна тачка. Под њиме су тестовима покривене следеће улазне тачке: поља форми за унос података (укључујући падајуће листе и поља за потврду) и колачићи (енгл. *cookie*). Код поља форми попут дугмади за обележавање (енгл. *checkbox*), корисник који има намеру да угрози апликацију може преко алата прегледача да измени вредност поља и за које му то није било омогућено.

На слици 4.1.1. је приказан изглед теста 18 из бенчмарка, који је позитиван на SQL инјекцију услед могућности уметања непроверене вредности кроз поље форме. Улазна вредност коју корисник пружа се директно уграђује као део упита за уметање у табелу *users*, што представља други случај код претходно поменутих ситуација у којима се ова претња испољава. Овде је лако извршити напад коришћењем надовезаног упита (енгл. *Piggy-backed Queries*), који је поменут у одељку 2.1.1, тако што корисник унесе за параметар вредност „foo’); DROP TABLE users;--“. На овај начин је кориснику омогућен покушај брисања табеле, као једне од могућности за напад.

Први корак аутора ових тестова је био да се упознају са природом интеракције апликације и сервера базе података, који као резултат омогућава приступ подацима. Примери такве комуникације укључују форме за аутентикацију, претраживаче (енгл. *Search engines*) и сајтове за електронску трговину [23]. Услед тога, може се закључити да је циљ имплементираних тестова у оквиру бенчмарка био да садрже различите комбинације поменутих атрибута, односно врсте упита и извора инјекције којим би се тестирао алат.

```

@WebServlet(value="/sqli-00/BenchmarkTest00018")
public class BenchmarkTest00018 extends HttpServlet {

    private static final Long serialVersionUID = 1L;

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // some code
        response.setContentType("text/html;charset=UTF-8");

        String param = "";
        java.util.Enumeration<String> headers = request.getHeaders("BenchmarkTest00018");

        if (headers != null && headers.hasMoreElements()) {
            param = headers.nextElement(); // just grab first element
        }

        // URL Decode the header value since req.getHeaders() doesn't. Unlike req.getParameters().
        param = java.net.URLDecoder.decode(param, "UTF-8");

        String sql = "INSERT INTO users (username, password) VALUES ('foo','" + param + "')";

        try {
            java.sql.Statement statement = org.owasp.benchmark.helpers.DatabaseHelper.getSqlStatement();
            int count = statement.executeUpdate( sql );
            org.owasp.benchmark.helpers.DatabaseHelper.outputUpdateComplete(sql, response);
        }
    }
}

```

Слика 4.1.1. Изглед теста бенчмарка позитивног на SQL инјекцију

У оквиру тестова који су наведени као негативни на SQL инјекцију у оквиру бенчмарка, уочено је коришћење објеката и принципа наведених под начинима заштите у одељку 2.1.1. Битно је истакнути да су у бенчмарку имплементиране додатне методе и класе које су енкапсулирале начин обраде података, где се у оквиру једне групе вршила валидација или враћање константне исправне вредности, а у другој само прослеђивање података без вршења адекватних провера. Пример такве класе је класа *SeparateClassRequest*, чији позив методе *getTheValue* враћа стринг „bar“ као константне вредности, док позив методе *getTheParameter* враћа неизмењен параметар онако како га је унео корисник [3]. Код ове класе је дат у прилогу рада (Прилог А.2).

4.2. Имплементација додатних тестова за SQL инјекцију

На основу дефинисане области коју покривају постојећи тестови бенчмарка у поглављу 4.1., овим радом је тај скуп проширен за десет тестова којим би се алати додатно проучили. Мотивација је била да се пронађу улазне тачке, тачке за напад на апликацију или њихове комбинације које нису биле укључене тестовима.

Уградња тестова наведених у наредним одељцима у оквиру бенчмарка захтевала је пружање фајлова написаних у програмском језику *Java* са кодом сервлета, XML фајлова са метаподацима који су претходно поменути у уводу поглавља четири и HTML фајлова који се односе на изглед страница за веб апликацију бенчмарка. У наставку су наведени кораци измене бенчмарка којим би се омогућило додавање теста, уз коришћење примера додавања теста „*BenchmarkTest02741*“:

- 1) У оквиру фајла „*expectedresults-1.2.csv*“ се додаје ред са информацијама о унетом тесту, односно за дефинисан пример се додаје ред „*BenchmarkTest02741,sqli,false,89*“. Овај ред укључује следеће запетом раздвојене

информације: назив теста, категорију претње којој тест припада, да ли је у питању права претња и CWE вредност.

- 2) У директоријум „*benchmark\src\main\java\org\owasp\benchmark\testcode*“ се додају фајлови у програмском језику *Java* и XML фајлови теста. На слици 4.2.1. је приказан садржај XML фајла за тест „*BenchmarkTest02741*“, који поред основних информација о самом тесту укључује и верзију коришћеног бенчмарка. Фајл са кодом теста „*BenchmarkTest02741.java*“ подразумева измену основног узорка по којем су базирани други тестови, како би се прилагодило потребама жељеног теста. Поменути узорак је приказан на слици 4.2.2., где је неопходно променити назив жељеног теста-сервлета и допунити код у оквиру методе *doPost*.

```
<test-metadata>
  <benchmark-version>1.2</benchmark-version>
  <category>sql</category>
  <test-number>02741</test-number>
  <vulnerability>false</vulnerability>
  <cwe>89</cwe>
</test-metadata>
```

Слика 4.2.1. Садржај фајла „*BenchmarkTest02741.xml*“

```
package org.owasp.benchmark.testcode;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(value="/sqli-07/BenchmarkTest02741")
public class BenchmarkTest02741 extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
        doPost(request, response);
    }

    @Override
    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException { /* OBRADA TESTA */
    }
}
```

Слика 4.2.2. Садржај дела фајла „*BenchmarkTest02741.java*“

- 3) У оквиру директоријума „*benchmark\src\main\webapp*“ се додаје HTML фајл, на који ће се указивати из фајла „*sqli-Index.html*“ кроз одговарајућу хипервезу. На слици 4.2.3. је приказан садржај фајла „*BenchmarkTest02741.html*“, где се од корисника захтева унос корисничког имена и лозинке у покренутој веб апликацији.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script src="/benchmark/js/jquery.min.js"></script>
<script type="text/javascript" src="/benchmark/js/js.cookie.js"></script>
<title>BenchmarkTest02741</title>
</head>
<body>
<form action="/benchmark/sqli-07/BenchmarkTest02741" method="GET" id="FormBenchmarkTest02741">
  <div><label>Login:</label></div>
  <br/>
  <div><label>Username:</label></div>
  <div><input type="text" id="username" name="username"></div>
  <div><label>Password:</label></div>
  <div><input type="text" id="password" name="password" value=""></div>
  <div>&nbsp;</div>
  <br/>
  <div><input type="submit" value="Login" /></div>
</form>
</body>
</html>
```

Слика 4.2.3. Садржај фајла „*BenchmarkTest02741.html*“

Након обављених корака, покретање генератора резулатата (енгл. *Scorecard*) и веб апликације бенчмарка ће укључити додати тест у оквиру свог статистичког прегледа и приказати валидан резултат уколико су алати покретани над новим скупом тестова.

У табели 4.2.1. приказане су основне информације везане за додатно имплементиране тестове-сервлете. У одељцима у наставку овог потпоглавља су детаљније наведене њихове дефиниције, где је за сваки тест је наведена вредност исхода везаног за претњу SQL инјекције.

Табела 4.2.1. Основне информације везане за додате тестове-сервлете

Назив теста (<i>BenchmarkTest0274x</i>)	Права претња	Упит	Напомене (садржај и потенцијална заштита)
1	Не	<i>"SELECT * FROM users WHERE username=? AND password=? LIMIT 1"</i>	Форма за пријављивање и заштита кроз параметризовани SQL упит
2	Не	<i>"SELECT * FROM " + param</i>	Опционо дугме за селекцију табеле за приказ + листа валидних улазних вредности
3	Не	<i>"DROP TABLE SCORE"</i>	Брисање табеле коришћењем дугмета за потврду

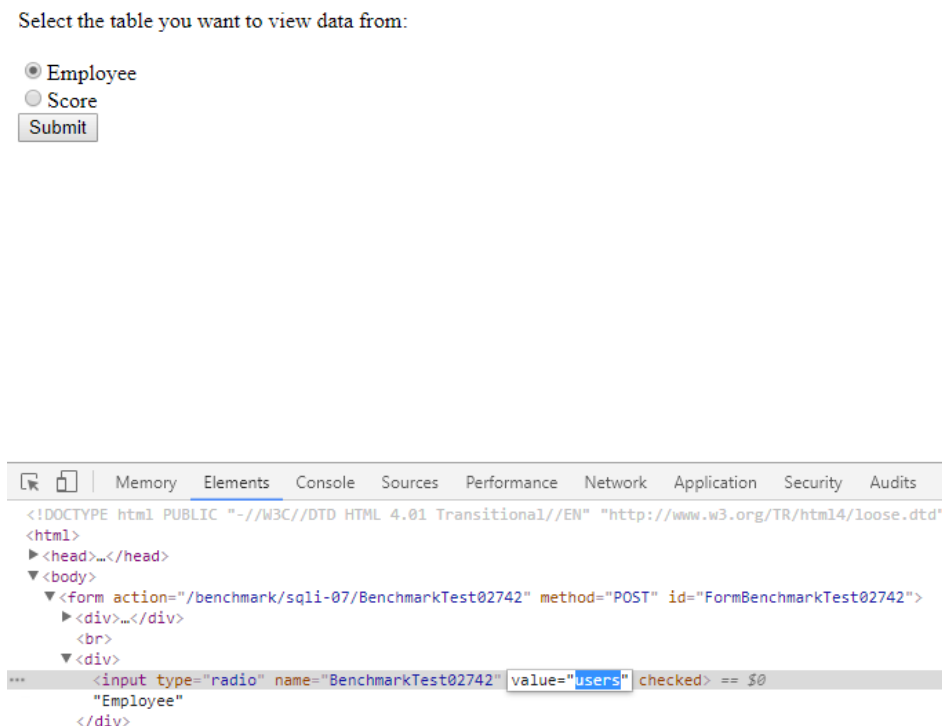
Назив теста (<i>BenchmarkTest0274x</i>)	Права претња	Упит	Напомене (садржај и потенцијална заштита)
4	Да	<i>"DELETE FROM score WHERE nick=" + param + ""</i>	Опционо дугме за брисање реда из табеле + <i>param.replace("", "")</i>
5	Да	<i>"DELETE FROM score WHERE nick=" + param + ""</i>	Опционо дугме за брисање реда из табеле + <i>org.apache.commons.lang.StringEscapeUtils</i>
6	Не	<i>"SELECT * FROM score WHERE userid=?"</i>	Приказ садржаја везан за унети целобројни атрибут + параметризовани SQL упит
7	Да	<i>"SELECT * FROM score WHERE userid=?"</i>	Приказ садржаја везан за унети целобројни атрибут + параметризовани SQL упит + испис грешке кориснику
8	Да	<i>"SELECT " + param + " FROM score"</i>	Дугмад за обележавање за селекцију више колона
9	Не	<i>"SELECT " + param + " FROM score"</i>	Дугмад за обележавање за селекцију више колона + провера на једнакост достављених вредности
10	Да	<i>"SELECT * FROM score WHERE score BETWEEN " + Integer.parseInt(startScore) + " AND " + Integer.parseInt(endScore)</i>	Унос целобројног опсега за селекцију садржаја за приказ + приказ упита у случају грешке у уносу

4.2.1. Тест „*BenchmarkTest02741*“

Тест „*BenchmarkTest02741*“ проверава унос корисничког имена и лозинке у оквиру форме за пријављивање коришћењем *java.sql.PreparedStatement* објекта, односно њене методе за постављање параметра типа стринг. Унети подаци нису валидирани и они се издвајају из захтева коришћењем мапе. Коришћен је упит изгледа „*SELECT * FROM users WHERE username=? AND password=? LIMIT 1*“ и с обзиром да је у питању параметризовани SQL упит, овај тест је негативан на SQL инјекцију. Овај тест се разликује од постојећих на основу коришћене комбинације пружања улазних података, начина њиховог преузимања користећи мапу и самог упита, где се вредности везују за обе колоне табеле *users*.

4.2.2. Тест „BenchmarkTest02742“

Тест „BenchmarkTest02742“ се односи на проверу начина одабира садржаја табеле за приказ селектовањем одговарајућег опционог дугмета (енгл. *radio button*). Покушај напада се може извршити изменом поља *value* одабраног опционог дугмета, који се неизмењен прослеђује апликацији у оквиру захтева.



Слика 4.2.2.1. Изглед теста „BenchmarkTest02742“ у оквиру веб апликације

У коду теста је коришћена новина у виду условне наредбе *IF*, којом се одабрана опција проверава и тиме се одређује која се табела приказује. Поред тога, упит није параметризован, већ је изгледа „*SELECT * FROM*“ на који се надовезује прослеђени параметар. Коришћењем начина валидације податка поређењем са листом валидних улазних вредности (енгл. *Whitelist*) поменутог у поглављу 2.1.1., овај тест је негативан на SQL инјекцију. Изглед овог теста у оквиру веб апликације бенчмарка је приказан на слици 4.2.2.1., где је на дну приказан поменути покушај напада. Уколико се унесе било која вредност осим вредности „*employee*“ и „*score*“, приказује се порука о грешци „*Error processing request.*“.

4.2.3. Тест „BenchmarkTest02743“

Тестом „BenchmarkTest02743“ се приказује коришћење нове врсте упита у односу на имплементиране, која се односи на брисање табеле из базе података. Брисање се реализује акцијом корисника спроведеном помоћу дугмета за потврду о брисању. Упит који се спроводи је изгледа „*DROP TABLE SCORE*“, који се реализује искључиво ако је прослеђена вредност „*Delete*“, што га чини негативним на SQL инјекцију.

4.2.4. Тест „BenchmarkTest02744“

Тест „BenchmarkTest02744“ приказује коришћење нове врсте упита у односу на имплементиране, која се односи на брисање реда из табеле који је везан за унуту вредност одговарајуће колоне. Од корисника се захтева избор оцене за брисање селекцијом опционог

дугмета, чија вредност садржи вредност колоне надимка „*nick*“. Пре извршавања упита изгледа „*DELETE FROM score WHERE nick='\" + param + \"'*“, унети параметар се мења једино ако дође до појаве апострофа у њему, при чему се он мења са два апострофа („*param.replace('\"', '\"')\"*“). Иако је тешко наћи пример којим би се потврдило да је овај тест позитиван на SQL инјекцију, он јесте позитиван и садржи потенцијалну опасност, јер се обрада унетих података од стране корисника ради заштите од специјалних односно небезбедних карактера не препоручује као вид заштите, што је поменуто у поглављу 2.1.1.

4.2.5. Тест „*BenchmarkTest02745*“

Тест „*BenchmarkTest02745*“ је сличан претходно поменутом тесту „*BenchmarkTest02744*“ у одељку 4.2.4., где је једина разлика у начину обраде унетог податка од стране корисника. За обраду је коришћена класа *org.apache.commons.lang.StringEscapeUtils*, која ручно решава двосмисленост карактера. Као и у претходном тесту, ова класа не представља потпуни начин заштите, јер она обезбеђује искључиво обраду карактера апострофа тиме што га претвара у два апострофа, услед чега је тест позитиван на SQL инјекцију [24].

4.2.6. Тест „*BenchmarkTest02746*“

Тестом „*BenchmarkTest02746*“ се проверава приказ одређеног реда из табеле *score* коришћењем *java.sql.PreparedStatement* објекта, односно њене методе за постављање параметра целобројног типа (енгл. *integer*) и упита изгледа „*SELECT * FROM score WHERE userid=?*“. Унос идентификатора корисника чије се информације приказују, врши се коришћењем текстуалног поља, који се доставља сервлету у оквиру стринга. Овај тест је негативан на SQL инјекцију, јер је коришћен параметризован упит, иако није извршена валидација прослеђеног параметра.

4.2.7. Тест „*BenchmarkTest02747*“

Тест „*BenchmarkTest02747*“ је сличан претходно поменутом тесту „*BenchmarkTest02746*“ у одељку 4.2.6., где је једина разлика у томе што се кроз грешку у уносу прослеђеног податка приказује SQL упит. Приказ грешке се везује за тип напада нелегалног тј. логички некоректног упита, којим унос вредности типа који није целобројна вредност условљава приказ упита, при чему је овај тест позитиван на SQL инјекцију.

4.2.8. Тест „*BenchmarkTest02748*“

Тестом „*BenchmarkTest02748*“ се проверава приказ података одређених колона из табеле *score*. Корисник бира одговарајуће колоне коришћењем дугмади за обележавање (енгл. *checkbox*), где се те вредности прослеђују упиту „*SELECT \" + param + \" FROM score*“ након њихове обраде. Сама обрада подразумева форматирање параметара за прослеђивање упиту, односно спајање параметара запетом уколико их је више користећи методу *String.join()*. Пошто се у овом случају не користи пре-компајлирана наредба поменута као вид заштите у поглављу 2.1.1., овај тест је позитиван на SQL инјекцију, јер се кроз параметре може извршити напад којим би се променио изглед упита.

4.2.9. Тест „*BenchmarkTest02749*“

Тест „*BenchmarkTest02749*“ је сличан претходно поменутом тесту „*BenchmarkTest02748*“ у одељку 4.2.8., где је једина разлика у томе што је употребљена провера унетих параметара коришћењем провере на једнакост достављених вредности са

исправним и очекиваним вредностима. На слици 4.2.9.1. је приказан део кода који се односи на обраду унетих вредности поређењем са исправним вредностима у низу стрингова *fields*. Резултат је тест који је негативан на SQL инјекцију.

```
String param = "";
String[] fields = {"userid", "nick", "score"};
for (int i = 0; i < values.length; i++) {
    if (!java.util.Arrays.asList(fields).contains(values[i])) {
        response.getWriter().println("Error processing request.");
        return;
    }
}

param = String.join(",", values);
sql = "SELECT " + param + " FROM score";

java.sql.Statement statement = org.owasp.benchmark.helpers.DatabaseHelper.getSqlStatement();
statement.execute(sql);
org.owasp.benchmark.helpers.DatabaseHelper.printResults(statement, sql, response);
```

Слика 4.2.9.1. Део кода теста „*BenchmarkTest02749*“

4.2.10. Тест „*BenchmarkTest02750*“

Последњи тест „*BenchmarkTest02750*“ уводи промену на тај начин што проверава алате кроз нову врсту упита, односно дела упита. Упит је изгледа „*SELECT * FROM score WHERE score BETWEEN " + Integer.parseInt(startScore) + " AND " + Integer.parseInt(endScore)*“, где је новина у провери опсега. Од корисника се захтева унос две вредности кроз текстуална поља, која се односе на вредност колоне резултата. У овом тесту не постоји валидација унетих вредности и кроз грешку у уносу прослеђеног податка, односно ако није унета целобројна вредност приказује се SQL упит. Овде је приказ упита условљен променљивом у оквиру дела кода за обраду изузетка. Додатна разлика у односу на постојеће тестове је што та променљива има тачну вредност, што значи да се приказује упит, услед чега је овај тест позитиван на SQL инјекцију.

5. Резултати

Примарни фокус овог поглавља је на поређењу добијених резултата након извршених анализа од стране алата који су описани у поглављу три. Обављене су анализе над два скупа тестова описаних у претходном поглављу, где први скуп не укључује додатне тестове за сигурносни пропуст SQL инјекције, како би се показало адекватно поређење између та два скупа.

Предуслов вршења поређења резултата је да су они изгенерисани помоћу генератора резултата (енгл. *Scorecard*), односно поменуте скрипте „*createScorecard.bat / .sh*“ која је позива. На овај начин се врши парсирање излазних XML фајлова алата из директоријума „*benchmark\results*“ и поређење са очекиваним резултатима бенчмарка. Резултат је садржај директоријума „*benchmark\scorecard*“, који наводи детаље поређења у виду слика, табела и веб страница. Поред детаља везаних за покривеност сигурносних претњи, изложени су детаљи који се односе на брзину извршавања односно време потребно за извршавање анализе. Међутим, оно се не може разматрати за све алате, јер немају сви алати опцију за чување времена о извршавању, што би довело до непрецизности у поређењу алата.

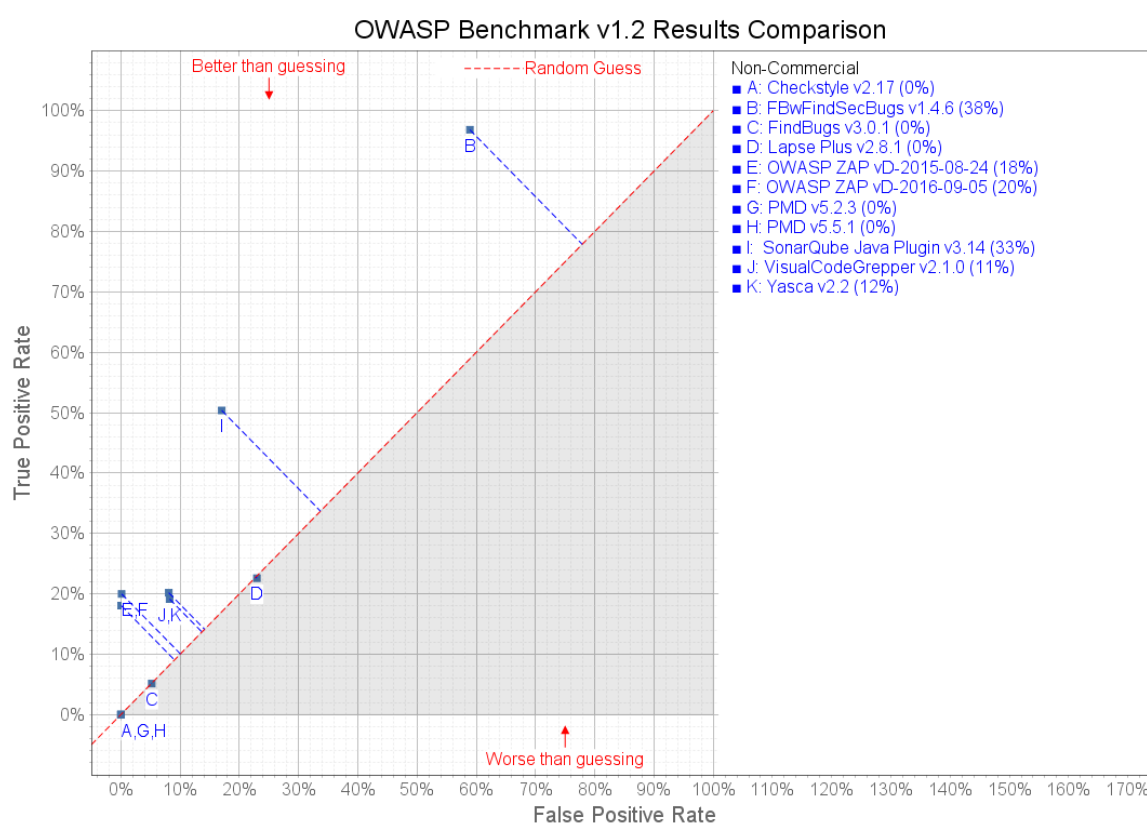
У наставку су раздвојени резултати у оквиру два потпоглавља, где се потпоглавље 5.1. односи на приказ поређења резултата анализе користећи примарни скуп тестова који је добијен са бенчмарком. Оно је укључило све сигурносне претње које бенчмарк подржава тестовима укључујући SQL инјекцију. Друго потпоглавље, односно потпоглавље 5.2., односи се на поређење прецизности алата које је везано за сигурносну претњу SQL инјекција, услед додатог подскупа тестова везаних за њу. Поред тога, детаљније су наведени разлози због којих су поједини тестови довели до одређених резултата, који су засновани на наведеним детаљима у имплементацији и генерисаним излазима алата. У потпоглављу 5.3. су наведена уочена запажања, предности и мане, која су везана за коришћене алате за статичку детекцију кода изведена из резултата у потпоглављима 5.1. и 5.2.

5.1. Приказ резултата анализе алата користећи тестове бенчмарка

Приказ резултата анализе алата користећи тестове бенчмарка се односи на резултате анализе које су извршили алати *PMD*, *FindBugs*, *FindBugs* са прикључком *FindSecBugs*, *Checkstyle*, *Lapse Plus*, *VisualCodeGrepper* и *Yasca*. На слици 5.1.1. је приказано визуелно поређење за све алате и тестове за све сигурносне претње, које је настало као производ генератора резултата. Као што је поменуто у поглављу четири, алат који детектује све претње и нема лажних детекција биће приказан у горњем левом углу графика. Десно од графика су у листи наведени алати чији су резултати поређени, укључујући и алат *OWASP ZAP* за динамичку анализу кода, који из тог разлога није разматран [3] и алат *SonarQube* за статичку анализу кода, чија анализа није разматрана у овом раду услед грешке поменуте у поглављу 3. За неке алате су наведени и резултати који су се односили на претходне верзије алата, што није уклоњено како би се уочио напредак у верзијама.

Од свих алата за статичку детекцију кода се као најуспешнији издвојио алат *FindBugs* са прикључком *FindSecBugs*, чија вредност резултата прецизности у оквиру бенчмарка (енгл. *Benchmark accuracy score*) износи 38 посто са стварно позитивном стопом од 96.84 посто и лажно позитивном стопом од 58.91 посто. Са друге стране, алати *FindBugs* и *Lapse Plus* су

према датом поређењу најмање ефикасни са негативним вредностима резулата прецизности од -0.07 и -0.38 посто. Иако ова два алата детектују праве претње, она имају већу стопу детекције лажних претњи, што је довело до негативних вредности резулата прецизности. Будући да су на слици 5.1.1. заокружене вредности резулата прецизности и да нису наведене вредности стопа, детаљније су приказана поређења резулата за све алате у оквиру табеле 5.1.1. из које су изостављени алати *OWASP ZAP* и *SonarQube*, пошто у овом раду они нису разматрани.



Слика 5.1.1. Поређење резулата алата у оквиру бенчмарка

Табела 5.1.1. Сигурносне претње у оквиру бенчмарка

	Алат	TPR	FPR	Резултат прецизности
1	<i>Checkstyle</i> (верзија 2.17)	0.00%	0.00%	0.00%
2	<i>FindBugs</i> са прикључком <i>FindSecBugs</i> (верзија 1.4.6)	96.84%	58.91%	37.93%
3	<i>FindBugs</i> (верзија 3.0.1)	5.12%	5.19%	-0.07%
4	<i>Lapse Plus</i> (верзија 2.8.1)	22.58%	22.96%	-0.38%
5	<i>PMD</i> (верзија 5.2.3)	0.00%	0.00%	0.00%
6	<i>PMD</i> (верзија 5.5.1)	0.00%	0.00%	0.00%

	Алат	TPR	FPR	Резултат прецизности
7	<i>VisualCodeGrepper</i> (верзија 2.1.0)	19.11%	8.22%	10.89%
8	<i>Yasca</i> (верзија 2.2)	20.16%	8.07%	12.09%

У наставку су укратко наведени резултати који су везани за све типове сигурносних претњи које бенчмарк покрива тестовима, како би се приказао њихов домен покривености сигурносних претњи. Такође је представљено табеларно поређење резултата алата везано за сваку претњу.

За претњу **инјекције команде**, једини алати који су евидентирали детекцију исте су били алати *FindBugs* са прикључком *FindSecBugs* и *Lapse Plus*, што се може уочити у табели 5.1.2. Пошто је циљ прикључка *FindSecBugs* да побољша анализу кода са аспекта сигурности, логично је да је покретање алата *FindBugs* са њим имало бољи резултат са вредношћу резултата прецизности од 11.20 посто, услед детекције свих правих претњи и 111 лажних претњи. Алат *Lapse Plus* је имао негативну вредност резултата прецизности -2.16 посто услед детекције већег броја лажних претњи у односу на праве претње.

Табела 5.1.2. Резултати анализе алата везани за инјекцију команде

	Алат	TPR	FPR	Резултат прецизности
1	<i>FindBugs</i> са прикључком <i>FindSecBugs</i> (верзија 1.4.6)	100.00%	88.80%	11.20%
2	<i>FindBugs</i> (верзија 3.0.1)	0.00%	0.00%	0.00%
3	<i>Lapse Plus</i> (верзија 2.8.1)	69.84%	72.00%	-2.16%

Код претњи **XSS** и **манипулације путањама** је слична ситуација (приказана у табели 5.1.3.) као код инјекције команде, са додатком детекције од стране алата *FindBugs* који је детектовао само једну XSS претњу и три праве и две лажне претње везане за манипулацију путањама. Једина разлика је у томе што код манипулације путањама алат *FindBugs* са прикључком *FindSecBugs* није детектовао све праве претње, због чега је његова вредност TPR износила 96.24 посто.

Табела 5.1.3. Резултати анализе алата везани за XSS и манипулацију путањама

	Алат	XSS			Манипулација путањама		
		TPR	FPR	Резултат прецизности	TPR	FPR	Резултат прецизности
1	<i>FindBugs</i> са прикључком <i>FindSecBugs</i> (верзија 1.4.6)	100.00%	62.68%	37.32%	96.24%	86.87%	9.57%
2	<i>FindBugs</i> (верзија 3.0.1)	0.41%	0.00%	0.41%	2.26%	1.48%	0.77%

	Алат	XSS			Манипулација путањама		
		TPR	FPR	Резултат прецизности	TPR	FPR	Резултат прецизности
3	<i>Lapse Plus</i> (верзија 2.8.1)	19.92%	22.01%	-2.09%	55.64%	59.26%	-3.62%

Детекцију претње везане за **колачић без постављеног атрибута за сигурност** (енгл. *Secure Cookie Flag*) је пружио само алат *FindBugs* са прикључком *FindSecBugs*, што је приказано у табели 5.1.4., који је спровео ефикасну анализу са детекцијом свих правих претњи и стопом FPR од само 12.90 посто. Са мало лошијим резултатом је био једини и за детекцију **LDAP инјекције и кршења граница поверења** (енгл. *Trust Boundary Violation*), где је детектовао све праве претње. Међутим, услед веће вредности стопе FPR имао је вредност резултата прецизности од 15.63 посто за LDAP инјекцију и 18.60 посто за претњу везану за кршење граница поверења, што се може уочити у табели 5.1.4.

Табела 5.1.4. Резултати анализе алата *FindBugs* са прикључком *FindSecBugs* везани за колачић без постављеног атрибута за сигурност, LDAP инјекцију и кршење граница поверења

<i>FindBugs</i> са прикључком <i>FindSecBugs</i>								
Колачић без постављеног атрибута за сигурност			LDAP инјекција			Кршење граница поверења		
TPR	FPR	Резултат прецизности	TPR	FPR	Резултат прецизности	TPR	FPR	Резултат прецизности
100.00%	12.90%	87.10%	100.00%	84.38%	15.63%	100.00%	81.40%	18.60%

Код претње везане за **слаб криптографски алгоритам** у табели 5.1.5. се издвајају алати *FindBugs* са прикључком *FindSecBugs*, *VisualCodeGrepper* и *Yasca*. Најефикаснији алат је био *Yasca*, који је успешно детектовао све праве претње са вредношћу резултата прецизности једнакој максималних 100 посто. Алат *FindBugs* са прикључком *FindSecBugs* је детектовао све праве претње, али и делом лажне због чега је његова вредност резултата прецизности износила 54.31 посто. Приближну вредност је имао и алат *VisualCodeGrepper* са 52.20 посто, али је он имао стопу TPR у износу од 74.62 посто, што значи да није пријавио све праве претње. Са друге стране, претња у истом домену која је везана за **употребу слабе хеш функције**, дефинише резултате за алате *FindBugs* са прикључком *FindSecBugs* и *Yasca* без детекције лажних претњи, али са вредностима резултата прецизности од 68.99 и 21.71 посто.

Претња везана за **употребу слабог генератора псеудослучајних вредности** издваја алат *FindBugs* са прикључком *FindSecBugs*, који је имао резултат од максималних 100 посто. Поред њега се показао успешним и алат *VisualCodeGrepper* са вредношћу резултата прецизности од 69.62 посто због недостатка детектованих правих претњи и мањег броја лажних претњи, што се уочава у табели 5.1.6. Трећи алат који је за ову претњу вршио анализу је алат *Yasca*, који је имао вредност резултата прецизности и TPR стопу једнаку 11.47 посто.

Табела 5.1.5. Резултати анализе алата везани за употребу слабог криптографског алгоритма и слабе хеш функције

	Алат	Слаб криптографски алгоритам			Слаба хеш функција		
		TPR	FPR	Резултат прецизности	TPR	FPR	Резултат прецизности
1	<i>FindBugs</i> са прикључком <i>FindSecBugs</i> (верзија 1.4.6)	100.00%	45.69%	54.31%	68.99%	00.00%	68.99%
2	<i>VisualCodeGrepper</i> (верзија 2.1.0)	74.62%	22.41%	52.20%	0.00%	0.00%	0.00%
3	<i>Yasca</i> (верзија 2.2)	100.00%	00.00%	100.00%	21.71%	0.00%	21.71%

Табела 5.1.6. Резултати анализе алата везани за употребу слабог генератора псеудослучајних вредности

	Алат	TPR	FPR	Резултат прецизности
1	<i>FindBugs</i> са прикључком <i>FindSecBugs</i> (верзија 1.4.6)	100.00%	00.00%	100.00%
2	<i>VisualCodeGrepper</i> (верзија 2.1.0)	88.53%	18.91%	69.62%
3	<i>Yasca</i> (верзија 2.2)	11.47%	0.00%	11.47%

XPATH инјекција је претња за коју су се алати показали најмање ефикасним, што је приказано у оквиру табеле 5.1.7. Алат *FindBugs* са прикључком *FindSecBugs* је детектовао све праве претње, али и већину лажних претњи, због чега је његова вредност резултата прецизности једнака пет посто. Поред њега, вредности стопа алата *Lapse Plus* су износиле 53.33 посто за TPR и 45.00 посто за FPR.

Табела 5.1.7. Резултати анализе алата везани за XPATH инјекцију

	Алат	TPR	FPR	Резултат прецизности
1	<i>FindBugs</i> са прикључком <i>FindSecBugs</i> (верзија 1.4.6)	100.00%	95.00%	5.00%
2	<i>Lapse Plus</i> (верзија 2.8.1)	53.33%	45.00%	8.33%

Једина претња за коју су сви алати имали подршку за анализу је **SQL инјекција**, осим алата *Checkstyle* и *PMD* који анализирају стил кодовања, што се може уочити у табели 5.1.8. Међутим, резултати алата су се показали таквим да једино алат *FindBugs* са прикључком *FindSecBugs* има позитивну вредност резултата прецизности са свим детектованим правим претњама, али и са 129 пријављених лажних претњи. Иако постоји свест о овој претњи, она представља велики проблем у оквиру домена сигурности, на шта сами резултати указују.

Значајна је чињеница да је алат за динамичку анализу кода *OWASP ZAP* (који није разматран) пружио висок проценат покривености са 34.13 посто, који се односи на старију верзију алата и 56.80 посто, који се односи на тренутну верзију. Може се формирати закључак на основу ових резултата да је за ефикаснију детекцију SQL инјекције додатно неопходан алат који пружа динамичку анализу кода попут алата *OWASP ZAP*. Разлог за то лежи у самим тестовима, где група њих садрже претње које се не могу детектовати на основу статичке анализе кода, услед чега је потребно да се тестови анализирају приликом извршавања истих. Типови и примери таквих тестова у оквиру бенчмарка су следећи:

- тестови чији ток извршавања зависи од условних наредби које се могу проверити тек у току извршавања (нпр. „*BenchmarkTest00104*“);
- тестови који употребљавају полиморфизам, где се одлука може донети само приликом извршавања (нпр. „*BenchmarkTest00107*“) и
- тестови који користе манипулације кроз колекције у програмском језику *Java* попут листи и хеш мапи или посебних метода у оквиру којих се смешта и дохвата вредност параметра за прослеђивање упиту (нпр. „*BenchmarkTest00197*“).

Табела 5.1.8. Резултати анализе алата везани за SQL инјекцију

	Алат	TPR	FPR	Резултат прецизности
1	<i>Checkstyle</i> (верзија 2.17)	0.00%	0.00%	0.00%
2	<i>FindBugs</i> са прикључком <i>FindSecBugs</i> (верзија 1.4.6)	100%	90.52%	9.48%
3	<i>FindBugs</i> (верзија 3.0.1)	53.68%	55.60%	-1.93%
4	<i>Lapse Plus</i> (верзија 2.8.1)	49.63%	54.31%	-4.68%
5	<i>PMD</i> (верзија 5.2.3)	0.00%	0.00%	0.00%
6	<i>PMD</i> (верзија 5.5.1)	0.00%	0.00%	0.00%
7	<i>VisualCodeGreppler</i> (верзија 2.1.0)	47.06%	49.14%	-2.08%
8	<i>Yasca</i> (верзија 2.2)	88.60%	88.79%	-0.19%

5.2. Приказ резултата анализе алата везане за SQL инјекцију

У овом потпоглављу су наведени резултати везани за анализу тестова који се односе на SQL инјекцију. Поред тога, образложени су разлози услед чега су претње детектоване на основу проучавања резултујућих фајлова који су алати произвели након анализе.

У табели 5.2.1. су приказани исходи извршавања анализе над новим подскупом од десет тестова, укључујући и вредности резултата анализе тог подскупа и целокупног скупа тестова везаног за SQL инјекцију. Колона резултат анализе нових тестова је израчуната на основу представљена четири исхода у табели (стварно позитивни, лажно негативни, стварно негативни и лажно позитивни исходи), док је резултат анализе свих тестова издвојен из статистике представљене бенчмарком након поновног покретања са додатим тестовима.

Табела 5.2.1. Резултати анализе алата везани за додате тестове за SQL инјекцију

	Алат	TP	FN	TN	FP	Резултат анализе нових тестова	Резултат анализе свих тестова
1	<i>Checkstyle</i>	0	5	5	0	0%	0%
2	<i>FindBugs</i> са прикључком <i>FindSecBugs</i>	3	2	3	2	20%	9.83%
3	<i>FindBugs</i>	3	2	4	1	40%	-1.06%
4	<i>Lapse Plus</i>	2	3	5	0	40%	-3.71%
5	<i>PMD</i>	0	5	5	0	0%	0%
7	<i>VisualCodeGrepper</i>	0	5	5	0	0%	-1.89%
8	<i>Yasca</i>	3	2	4	1	40%	0.74%

Алати *Checkstyle* и *PMD* су имали очекивани резултат од нула посто, будући да су то алати за проверу изгледа изворног кода према стандарду и дефинисаним правилима, што је описано у поглављу три. Тестови „*BenchmarkTest02744*“ и „*BenchmarkTest02745*“ су успешно прошли код свих алата, изузев код оних са нула посто детекције. Ниједан алат није успео да детектује све праве претње, односно највећи проблем је постојао за тест „*BenchmarkTest02747*“. Разлог услед којег је овај тест представљао изазов лежи у чињеници да статички алати немају наведено правило за уочавање проблема приказа грешке. На исти начин је тест „*BenchmarkTest02750*“ поставио ситуацију у којој статички алати нису могли да утврде да је приказ грешке условљен променљивом чија је вредност тачна. Случајем околности је алат *Yasca* дефинисао овај тест као позитиван на SQL инјекцију, будући да има дефинисано правило приказано на слици 3.6.2. по којем наводи да је у питању права претња ако се користи непараметризован упит.

Како би се уочили недостаци алата при извршеној анализи, у наредним одељцима су наведена образложења грешака појединачних алата изузев алата *Checkstyle* и *PMD*. Уколико се разматрају сви тестови, алат *FindBugs* са прикључком *FindSecBugs* је имао најбољи резултат, али за креирани подскуп тестова је имао упола лошији резултат у односу на већину алата. Разлог за то је што су управо ови тестови обухватили примере, односно функције, које он сматра потенцијално небезбедним. У потпоглављу 5.3. је дискутовано како овај случај може утицати на ефикасност алата, као и ефикасност осталих коришћених алата.

5.2.1. Резултат анализе алата *VisualCodeGrepper* везан за SQL инјекцију

Алат *VisualCodeGrepper* имао исти резултат као *Checkstyle* и *PMD*, али из разлога што овај алат наводи детекцију претње везане за SQL инјекцију у случајевима приказаним на слици 3.5.1. У тестовима „*BenchmarkTest02744*“ и „*BenchmarkTest02745*“ су пре самог динамичког упита извршене обраде параметара које се обрађују у упиту, услед чега претње у овим тестовима нису препознате од стране алата. Овај случај указује на пропуст у алату који је неопходно обрадити.

5.2.2. Резултати анализе алата FindBugs и FindBugs са прикључком FindSecBugs везаних за SQL инјекцију

Алат *FindBugs* са прикључком *FindSecBugs* је лажно прогласио да тест „*BenchmarkTest02742*“ има претњу. Разлог због којег је то наведено код првог наведеног алата је услед уочене функције *Statement.execute(...)*, која је препозната код овог алата као потенцијална претња. Са друге стране, тест „*BenchmarkTest02749*“ су оба алата навела као позитиван на SQL инјекцију. Код алата *FindBugs* је то наведено услед прослеђивања стринга који није константа, тј. прослеђивања непараметризованог динамичког упита методи за његово извршавање. Често овакав случај указује на праву претњу, али може да постоји адекватна обрада параметра који се уграђује у стрингу којим се штити од ове претње. Са друге стране, алат *FindBugs* са прикључком *FindSecBugs* на исти начин дефинише ову претњу као и код теста „*BenchmarkTest02742*“. Употреба ове функције и интерфејса се не препоручује услед потенцијалне појаве претње, али постоји могућност да она није присутна услед адекватно пружене заштите, што је претходно наведено и за алат *FindBugs* са прикључком *FindSecBugs*.

5.2.3. Резултат анализе алата Yasca везан за SQL инјекцију

Алат *Yasca* је у тесту „*BenchmarkTest02742*“ навео као претњу појаву променљиве у оквиру упита због дефинисаног регуларног израза „*/\"(select|delete) .*from .*|+/i*“ приказаног на слици 3.6.2., који се поклопио са упитом коришћеним у овом тесту. Овај регуларни израз је због недостатка специфичности обухватио и овај случај, што указује на недостатак овог алата. Поред тога, праву претњу у оквиру теста „*BenchmarkTest02748*“ није успео да детектује. Према његовим правилима тј. регуларним изразима, уграђивање променљиве се детектује као претња ако је наведена искључиво након кључне речи *from*. У овом тесту се променљива уграђује након кључне речи *select*, односно променљивом се дефинишу колоне које се желе дохватити из табеле. Овај случај приказује недостатак у дефинисаним правилима који је неопходно допунити.

5.2.4. Резултат анализе алата Lapse Plus везан за SQL инјекцију

Овај алат није детектовао претњу у тесту „*BenchmarkTest02748*“, јер детекцију претњи своди на проналажење небезбедних функција, што је образложено у поглављу 3.3.

5.3. Анализа алата заснована на резултатима

Према резултатима наведеним у претходним поглављима, алат *FindBugs* са прикључком *FindSecBugs* се статистички показао најпожељнијим за употребу за детекцију свих типова сигурносних претњи. Међутим, може да се донесе закључак везан за детекцију свих сигурносних претњи, који се односи на неопходно стање извршавања за многе ситуације у којима се може вршити анализа кода. Оно се огледа у највећем недостатку статичких алата, који је представљен одликама несигурности појаве претње и немогућности доказивања да је у питању права претња. Овај случај се појављује приликом детекције SQL инјекције и код поменутог најефикаснијег алата *FindBugs* са прикључком *FindSecBugs*, који аутоматски проглашава претње као праве уколико се користи функција *Statement.executeUpdate()* или се прослеђује упиту стринг који није константа, што пријављује поруком о потенцијалној претњи. Наведени недостатак је индиректно повезан са високим бројем лажно позитивних исхода, који представља број лажних претњи које су детектоване као праве претње од стране алата. Уколико је овај број висок, ефикасност алата је знатно мања упркос потенцијалном проналаску свих правих претњи.

Из претходно поменутих резултата се може уочити да алат *FindBugs* извршава бољу анализу кода приликом тражења сигурносних претњи када укључује прикључак *FindSecBugs*. Том приликом, он има уграђену подршку за детекцију свих претњи из листе претњи бенчмарка. Ово запажање је логично будући да је намена додатог прикључка да се фокусира на претње у домену сигурности и на тај да побољша ефикасност алата. Такође је уочено да овај алат без додатог прикључка има најлошије свеукупне резултате у односу на остале тестиране алате (не укључујући алате *Checkstyle* и *PMD*).

Алат *Lapse Plus* у односу на алате *VisualCodeGrepper* и *Yasca* подржава анализу више врста сигурносних претњи, али има лошије резултате који су негативни за све осим ХРАТН инјекције (8.33 посто). Са друге стране, алат *VisualCodeGrepper* детектује коришћење слабог криптографског алгорита, слабог генератора псеудослучајних вредности и SQL инјекцију, где за прве две претње има прецизност преко 50 посто. У поређењу са њим, додатну претњу коришћења слабе хеш функције детектује алат *Yasca*. Међутим, ако се посматра само SQL инјекција, оба алата имају негативне вредности резултата прецизности, али је за два посто бољи алат *Yasca*.

Уколико се посматра надоградња алата, односно додавање правила од стране корисника, сви алати изузев алата *Lapse Plus* пружају ову могућност, што је описано у поглављу 3. Међутим, најједноставније начине за додавање правила пружају алати *Yasca* и *VisualCodeGrepper* кроз навођење регуларних израза и правила, док алати *Checkstyle*, *PMD* и *FindBugs* захтевају дефинисање посебних модула и детектора кроз која се дефинишу правила.

Упркос томе што алати *Checkstyle* и *PMD* нису пружили детекцију наведених претњи, њих не треба занемарити. Они пружају начин за побољшање писања кода према стандарду програмског језика, који индиректно утиче на аспект сигурности. Коришћењем ових алата се усмерава пажња на писање „доброг кода“ и на тај начин програмер учи да користи алате за превенцију грешака и документацију језика у којој се наводе безбедне и пожељне функције и објекти за употребу.

Може се закључити да се за анализу апликације која захтева детаљан и крајње ефикасан проналазак сигурносних претњи не треба у потпуности ослонити само на алате за статичку детекцију кода. Они треба да буду употребљени приликом развоја апликације, али треба да постоје и одговарајуће фазе тестирања које укључују друге методологије и врсте алата које би биле комплементарне са овим. Конкретно за ефикаснију детекцију претње SQL инјекције је неопходан додатни алат који пружа динамичку анализу кода, будући да се многе претње не могу уочити уколико није апликација у стању извршавања. Ова чињеница је веома битна за бољу анализу кода везану за SQL инјекцију, али је пожељна и за друге претње. Поред коришћења различитих алата, у већини ситуација је такође потребна и мануелна валидација.

Иако постоје недостаци везани за ову врсту алата, могу се уочити и битне предности које су од есенцијалног значаја за развој безбедних апликација. Њих чине следеће предности:

- Приказ резултата анализе који укључује опис претње и део кода где је уочена;
- Није потребно да буде апликација у стању извршавања, што значи да се може извршити анализа пре фазе испоруке (енгл. *deployment*), односно чак и у почетним фазама развоја;
- Добра скалабилност;
- Лака детекција грешака приликом кодирања;
- Једноставна интеграција и

- Детаљно проучавање изворног кода (потенцијални недостатак уколико није доступан).

На крају треба истаћи да је напредак алата за статичку анализу кода видљив кроз поређења резултата прецизности претходних и тренутних верзија алата, што указује на напор који се пружа за креирање доброг алата. Према резултатима наведеним у овом раду, може да се издвоји алат *FindBugs* са прикључком *FindSecBugs* као тренутно најефикаснији алат за статичку анализу кода у домену сигурносних претњи, док су се остали алати за специфичне намене показали адекватним за употребу. Битно је да се прати развој ових алата, јер они могу да утичу на многе аспекте сигурности и да буду од великог значаја при креирању апликација.

6. Закључак

Примарни циљ овог рада је било испитивање ефикасности постојећих алата за статичку анализу кода, које се односило на детекцију SQLi сигурносних пропуста у апликацијама. Коришћењем OWASP бенчмарка, испитивање је проширено на скуп различитих сигурносних претњи, које се односе на највеће изазове у домену информационе безбедности. На тај начин, изведени су закључци везани за коришћене алате поређењем њихове покривености и прецизности. Поред тога, детаљно је проучен бенчмарк кроз евалуацију комплексности интеграције различитих алата и тестова у оквиру његовог кода.

Приликом саме реализације процеса анализе, првенствено је било неопходно пронаћи одговарајуће алате за статичку анализу кода које би подржале интеграцију у оквиру бенчмарка. Том приликом је уочено да је употреба OWASP бенчмарка олакшала крајње поређење алата кроз омогућене статистичке евалуације и визуелне индикаторе, којима су издвојени ефикасни алати. Кроз добијене резултате, изведен је закључак да је алат *FindBugs* са прикључком *FindSecBugs* тренутно најефикаснији алат за статичку анализу кода у домену сигурносних претњи, укључујући SQLi сигурносни пропуст. Овај алат је доказао да прати појаву нових сигурносних претњи кроз подршку детекције свих претњи из листе претњи бенчмарка, што је битно услед константне појаве нових претњи и потраге за новим начинима за напад на апликације.

Са друге стране, испитан је начин детекције сигурносне претње SQLi кроз постојеће тестове бенчмарка и правила које чине алате за статичку анализу кода. Ово је омогућило надоградњу тренутног скупа тестова, којим су проверени додатни аспекти поменуте претње од стране алата и комплексност интеграције тестова у оквиру бенчмарка.

Разматране су могуће надоградње, које су произашле из анализе бенчмарка описане у поглављу два, као и у поглављима три и четири везаним за интегрисане алате и тестове. Први тип надоградње који би допринео широј примени бенчмарка је проширење врсте и комплексности тестова, јер тренутна верзија бенчмарка омогућава искључиво проверу кода у оквиру сервлета. Ова измена би представљала битан корак ка тестирању сложенијег индустријског софтвера.

Други тип надоградње се односи на алате за статичку анализу кода, где би од великог значаја било увођење интуитивнијег приказа пронађених дефеката и начина за њихово елиминисање, уколико је оно могуће. Алати који су у овом раду тестирани у већини случајева не пружају јасан и детаљан опис дефекта, чиме се троши време на тражење потенцијалне дефиниције дефекта у документацији. Такође искључивање појединих детектора у оквиру конфигурације алата би представљао значајно олакшање при тестирању одређеног подскупа претњи.

Извршеним проучавањем су потврђени једноставност и свестраност примене OWASP бенчмарка за сигурносну аутоматизацију. Иако он тренутно омогућава интеграцију за одређени скуп алата и тестова написаних искључиво у програмском језику *Java*, његов покушај представља значајан корак у евалуацији алата и апликација. Поред самог бенчмарка, напредак је уочен и код самих алата за статичку анализу кода, који теже све већој покривености скупа сигурносних претњи. Услед тога, спроведена анализа у овом документу

се може искористити као основа у будућем развоју и предузимању сигурносних мера у заштити апликација.

ЛИТЕРАТУРА

- [1] J. Witschey et al., "Technical and personal factors influencing developers' adoption of security tools", in *Proceedings of the 2014 ACM Workshop on Security Information Workers*. 2014, pp. 23–26.
- [2] OWASP [Online]. Available: https://www.owasp.org/index.php/Main_Page (17.07.2017.)
- [3] OWASP benchmark [Online]. Available: <https://www.owasp.org/index.php/Benchmark#tab=Main> (18.07.2017.)
- [4] OWASP Foundation. (2007). *The ten most critical web application security vulnerabilities for Java enterprise applications* [Online]. Available: https://www.owasp.org/images/8/89/OWASP_Top_10_2007_for_JEE.pdf (18.07.2017.)
- [5] *Bobby Tables: A guide to preventing SQL injection* [Online]. Available: <http://bobby-tables.com/> (18.07.2017.)
- [6] W. Halfond et al., "A classification of SQL-injection attacks and countermeasures", in *Proceedings of the IEEE International Symposium on Secure Software Engineering Vol. 1*. IEEE, 2006.
- [7] R. Chandrashekhar et al., "SQL Injection Attack Mechanisms and Prevention Techniques", in *Advanced Computing, Networking and Security: International Conference ADCONS*, India, 2011, pp. 526-527.
- [8] *Acunetix: Types of SQL Injection (SQLi)* [Online]. Available: <https://www.acunetix.com/websitesecurity/sql-injection2/> (20.07.2017.)
- [9] *Git* [Online]. Available: <https://git-scm.com/> (21.07.2017.)
- [10] *Apache Maven* [Online]. Available: <https://maven.apache.org/> (21.07.2017.)
- [11] *Java SE* [Online]. Available: <http://www.oracle.com/technetwork/java/javase/downloads/index.html> (21.07.2017.)
- [12] P. Emanuelsson and U. Nilsson, "A Comparative Study of Industrial Static Analysis Tools", *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 217. July, 2008.
- [13] *Checkstyle* [Online]. Available: <http://checkstyle.sourceforge.net/> (24.07.2017.)
- [14] C. Gotz, "Vulnerability identification in web applications through static analysis", M.S. thesis, Fakultät für Informatik der Technischen Univ. München, Germany, 2013.
- [15] *FindBugs* [Online]. Available: <http://findbugs.sourceforge.net> (25.07.2017.)
- [16] *Find Security Bugs* [Online]. Available: <http://find-sec-bugs.github.io/> (26.07.2017.)
- [17] *Lapse-plus Google Code* [Online]. Available: <https://code.google.com/archive/p/lapse-plus/downloads> (27.07.2017.)
- [18] *Lapse-plus* [Online]. Available: <https://github.com/bergerbd/lapse-plus> (27.07.2017.)
- [19] *PMD* [Online]. Available: <https://pmd.github.io/pmd-5.8.1/> (28.07.2017.)
- [20] *VisualCodeGrepper* [Online]. Available: <https://github.com/nccgroup/VCG> (28.07.2017.)
- [21] *Yasca User's Guide* [Online]. Available:

<http://web.archive.org/web/20150513154618/http://www.scovetta.com:80/yasca/yasca-manual.pdf> (29.07.2017.)

[22] *Youden's J statistic* [Online]. Available:

https://en.wikipedia.org/wiki/Youden%27s_J_statistic (17.07.2017.)

[23] *OWASP: Testing for SQL Injection* [Online]. Available:

[https://www.owasp.org/index.php/Testing_for_SQL_Injection_\(OTG-INPVAL-005\)](https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005))
(03.08.2017.)

[24] *StringEscapeUtils* [Online]. Available:

[https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang/StringEscapeUtils.html#escapeSql\(java.lang.String\)](https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang/StringEscapeUtils.html#escapeSql(java.lang.String))
(03.08.2017.)

СПИСАК СКРАЋЕНИЦА

AJAX	<i>Asynchronous JavaScript And XML</i>
API	<i>Application Programming Interface</i>
CSV	<i>Comma-separated values</i>
CWE	<i>Common Weakness Enumeration</i>
DNS	<i>Domain Name System</i>
DoS	<i>Denial of Service</i>
FN	<i>False Negative</i>
FP	<i>False Positive</i>
FPR	<i>False Positive Rate</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
JavaCC	<i>Java Compiler Compiler</i>
JDBC	<i>Java Database Connectivity</i>
JPA	<i>Java Persistence API</i>
JPQL	<i>Java Persistence Query Language</i>
LDAP	<i>Lightweight Directory Access Protocol</i>
OWASP	<i>The Open Web Application Security Project</i>
SAST	<i>Source code analysis tools / Static Application Security Testing Tools</i>
SQL	<i>Structured Query Language</i>
SQLi	<i>SQL injection</i>
TN	<i>True Negative</i>
TP	<i>True Positive</i>
TPR	<i>True Positive Rate</i>
XML	<i>Extensible Markup Language</i>
XPATH	<i>XML Path Language</i>
XSS	<i>Cross-Site Scripting</i>

СПИСАК СЛИКА

Слика 2.1.1. Пример рањивости на SQLi.....	5
Слика 2.1.2. Употреба <i>java.sql.PreparedStatement</i> објекта.....	8
Слика 2.1.3. Употреба ускладиштене процедуре.....	8
Слика 2.2.1. Команде за покретање бенчмарка у оквиру <i>Git Bash</i>	9
Слика 2.2.2. Део садржаја фајла „ <i>pom.xml</i> “.....	10
Слика 2.2.3. Почетна страница веб апликације бенчмарка.....	11
Слика 3.1.1. Дијаграм рада алата <i>Checkstyle</i>	14
Слика 3.1.2. Елемент за <i>Checkstyle</i> извештаје у конфигурационом фајлу „ <i>pom.xml</i> “.....	16
Слика 3.1.3. Команде за покретање алата <i>Checkstyle</i>	16
Слика 3.2.1. Дијаграм рада алата <i>FindBugs</i>	17
Слика 3.2.2. Команде за покретање алата <i>FindBugs</i>	18
Слика 3.2.3. Команде за покретање алата <i>FindBugs</i> са прикључком <i>FindSecBugs</i>	18
Слика 3.3.1. Прегледи корака алата <i>LAPSE+</i> у оквиру окружења <i>Eclipse</i>	20
Слика 3.3.2. Дијаграм процеса детекције тачке извршавања претње алата <i>LAPSE+</i>	20
Слика 3.3.3. Приказ статистике алата <i>LAPSE+</i>	21
Слика 3.4.1. Дијаграм рада алата <i>PMD</i>	22
Слика 3.4.2. Елемент за <i>PMD</i> извештаје у фајлу „ <i>pom.xml</i> “.....	23
Слика 3.4.3. Команде за покретање алата <i>PMD</i>	23
Слика 3.5.1. Дијаграм рада алата <i>VisualCodeGrepper</i>	24
Слика 3.5.2. Део кода алата <i>VisualCodeGrepper</i> за детекцију SQL инјекције.....	26
Слика 3.5.3. Преглед анализираног кода за алат <i>VisualCodeGrepper</i>	26
Слика 3.5.4. Преглед резултата анализе за алат <i>VisualCodeGrepper</i>	27
Слика 3.6.1. Дијаграм рада алата <i>Yasca</i>	28
Слика 3.6.2. Регуларни изрази за SQLi алата <i>Yasca</i>	29
Слика 3.6.3. Преглед опција приликом покретања алата <i>Yasca</i>	29
Слика 4.2. Визуелни индикатор успешности алата [3].....	31
Слика 4.1.1. Изглед теста бенчмарка позитивног на SQL инјекцију.....	33
Слика 4.2.1. Садржај фајла „ <i>BenchmarkTest02741.xml</i> “.....	34
Слика 4.2.2. Садржај дела фајла „ <i>BenchmarkTest02741.java</i> “.....	34
Слика 4.2.3. Садржај фајла „ <i>BenchmarkTest02741.html</i> “.....	35
Слика 4.2.2.1. Изглед теста „ <i>BenchmarkTest02742</i> “ у оквиру веб апликације.....	37
Слика 4.2.9.1. Део кода теста „ <i>BenchmarkTest02749</i> “.....	39
Слика 5.1.1. Поређење резултата алата у оквиру бенчмарка.....	41

СПИСАК ТАБЕЛА

Табела 2.1.1. Сигурносне претње у оквиру бенчмарка	4
Табела 3.1. Основне информације везане за коришћене алате.....	13
Табела 4.2.1. Основне информације везане за додате тестове-сервлете	35
Табела 5.1.1. Сигурносне претње у оквиру бенчмарка	41
Табела 5.1.2. Резултати анализе алата везани за инјекцију команде	42
Табела 5.1.3. Резултати анализе алата везани за XSS и манипулацију путањама	42
Табела 5.1.4. Резултати анализе алата <i>FindBugs</i> са прикључком <i>FindSecBugs</i> везани за колачић без постављеног атрибута за сигурност, LDAP инјекцију и кршење граница поверења	43
Табела 5.1.5. Резултати анализе алата везани за употребу слабог криптографског алгоритма и слабе хеш функције.....	44
Табела 5.1.6. Резултати анализе алата везани за употребу слабог генератора псеудослучајних вредности	44
Табела 5.1.7. Резултати анализе алата везани за XPATH инјекцију	44
Табела 5.1.8. Резултати анализе алата везани за SQL инјекцију	45
Табела 5.2.1. Резултати анализе алата везани за додате тестове за SQL инјекцију	46

A. Додатни програмски кôдови

У овом прилогу су дати садржаји фајлова у програмском језику *Java* који су претходно поменути у раду.

A.1. КЛАСА *ToXML*

У наставку је дат кôд класе *ToXML*, која представља конвертор излаза текстуалног фајла алата *Lapse Plus* у одговарајући XML фајл. Ова класа је била неопходан корак како би се успешно приказали и интегрисали резултати анализе у оквиру бенчмарка.

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.AttributesImpl;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.transform.sax.*;

public class ToXML {
    private BufferedReader in;
    private StreamResult out;
    TransformerHandler th;

    public static void main(String[] args) {
        new ToXML().begin();
    }

    public void begin() {
        String str = "";
        try {
            // UCITAVANJE TEKSTUALNOG FAJLA ZA KONVERZIJU
            in = new BufferedReader(new FileReader("konv.txt"));
            out = new StreamResult("konv.xml");
            openXml();

            while ((str = in.readLine()) != null) {
                String checktest = str.substring(136);
                if (checktest.startsWith(
"/benchmark/src/main/java/org/owasp/benchmark/testcode/BenchmarkTest"))
                    process(str);
            }

            in.close();
            closeXml();

        } catch (Exception e) {
```

```

        System.out.println(str);
        e.printStackTrace();
    }
}

/***** PODESAVANJE IZGLEDA XML FAJLA *****/
/*****
public void openXml() throws ParserConfigurationException,
TransformerConfigurationException, SAXException {

    SAXTransformerFactory tf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
    th = tf.newTransformerHandler();

    Transformer serializer = th.getTransformer();
    serializer.setOutputProperty("{http://xml.apache.org/xslt}indent-
amount", "4");
    serializer.setOutputProperty(OutputKeys.INDENT, "yes");

    th.setResult(out);
    th.startDocument();
    AttributesImpl atts = new AttributesImpl();
    atts.addAttribute("", "", "version", "", "2.8.1");
    th.startElement(null, null, "LapsePlusXMLResults", atts);
}

/***** PROCESIRANJE ELEMENATA XML FAJLA *****/
/*****
public void process(String s) throws SAXException {
    th.startElement(null, null, "result", null);

    th.startElement(null, null, "source", null);
    th.characters(s.toCharArray(), 0, 74);
    th.endElement(null, null, "source");

    th.startElement(null, null, "source_context", null);
    th.characters(s.toCharArray(), 74, 41);
    th.endElement(null, null, "source_context");

    th.startElement(null, null, "filename", null);
    th.characters(s.toCharArray(), 136+54, s.length() - (136+54));
    th.endElement(null, null, "filename");

    th.startElement(null, null, "testnumber", null);
    th.characters(s.toCharArray(), 203, 5);
    th.endElement(null, null, "testnumber");

    // OBRADA TIPa SIGURNOSNE PRETNJE
    // 1. INJEKCIJA KOMANDE
    if (s.startsWith("UNKNOWN r.exec")) {
        th.startElement(null, null, "category", null);

```

```

        th.characters("Command Injection".toCharArray(), 0, 17);
        th.endElement(null, null, "category");

        th.startElement(null, null, "cwe", null);
        th.characters("78".toCharArray(), 0, 2);
        th.endElement(null, null, "cwe");
    }
    // 2. XSS
    else if (s.startsWith("UNKNOWN response.getWriter()")) {
        th.startElement(null, null, "category", null);
        th.characters("XSS".toCharArray(), 0, 3);
        th.endElement(null, null, "category");

        th.startElement(null, null, "cwe", null);
        th.characters("79".toCharArray(), 0, 2);
        th.endElement(null, null, "cwe");
    }
    // 3. XPATH INJEKCIJA
    else if (s.startsWith("UNKNOWN xp.compile(expression)")) {
        th.startElement(null, null, "category", null);
        th.characters("XPATH Injection".toCharArray(), 0, 15);
        th.endElement(null, null, "category");

        th.startElement(null, null, "cwe", null);
        th.characters("643".toCharArray(), 0, 3);
        th.endElement(null, null, "cwe");
    }
    // 4. SQL INJEKCIJA
    else if (s.startsWith("UNKNOWN connection") ||
        s.startsWith("UNKNOWN statement") ||
        s.startsWith("UNKNOWN stmt")) {
        th.startElement(null, null, "category", null);
        th.characters("SQL Injection".toCharArray(), 0, 13);
        th.endElement(null, null, "category");

        th.startElement(null, null, "cwe", null);
        th.characters("89".toCharArray(), 0, 2);
        th.endElement(null, null, "cwe");
    }
    // 5. OBILAZAK PUTANJE
    else {
        th.startElement(null, null, "category", null);
        th.characters("Path Traversal".toCharArray(), 0, 14);
        th.endElement(null, null, "category");

        th.startElement(null, null, "cwe", null);
        th.characters("22".toCharArray(), 0, 2);
        th.endElement(null, null, "cwe");
    }

    th.endElement(null, null, "result");
}

```

```

    public void closeXml() throws SAXException {
        th.endElement(null, null, "LapsePlusXMLResults");
        th.endDocument();
    }
}

```

A.2. Kôd klase *SEPARATECLASSREQUEST* [3]

У наставку је дат кôд класе *SeparateClassRequest*, која је употребљена у оквиру тестова бенчмарка. Метода *getTheValue* враћа стринг „bar“ као константну вредност, док метода *getTheParameter* враћа неизмењен параметар онако како га је унео корисник. Ова класа је битна јер сигурност апликације зависи од енкапсулираног начина обраде улазних параметара.

```

package org.owasp.benchmark.helpers;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;

public class SeparateClassRequest {
    private HttpServletRequest request;

    public SeparateClassRequest( HttpServletRequest request ) {
        this.request = request;
    }

    public String getTheParameter(String p) {
        return request.getParameter(p);
    }

    // PRAVA PRETNJA (jer zavisi od ulaznih parametara zahteva)
    public String getTheCookie(String c) {
        Cookie[] cookies = request.getCookies();
        String value = "";

        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals(c)) {
                    value = cookie.getValue();
                    break;
                }
            }
        }

        return value;
    }

    // LAZNA PRETNJA
    public String getTheValue(String p) {
        return "bar";
    }
}

```

}

В. Додатна образложења сигурносних претњи

У оквиру овог прилога су дата додатна образложења поменутих сигурносних претњи у овом раду која нису детаљно проучавана као претња SQLi. Она су наведена у оквиру описа алата у поглављу три приликом навођења сигурносних претњи које детектују. Дефиниције су засноване на коду бенчмарка и коришћеној литератури, од којих је примарна OWASP документација [3].

В.1. ИНЈЕКЦИЈА КОМАНДЕ (ЕНГЛ. *COMMAND INJECTION*)

Инјекција команде је претња која је последица дозвољеног уноса и обраде непроверених улазних података пружених од стране корисника у оквиру љуске (енгл. *System Shell*). Корисници уносе злонамерне податке кроз улазне тачке попут форми и колачића, који се након тога не валидирају адекватно у оквиру апликације. На овај начин, нападач може да добије посебне привилегије, које му омогућавају извршавање команди на систему хоста.

В.2. СЛАБ КРИПТОГРАФСКИ АЛГОРИТАМ (ЕНГЛ. *WEAK CRYPTOGRAPHY*)

Употреба слабог криптографског алгорита се односи на коришћење небезбедних имплементација, односно небезбедних алгорита за заштиту података апликације који нарушавају њихов интегритет и тајност. У већини случајева су такви алгоритми означени као застарели и не препоручљиви у оквиру документација коришћених програмских језика. Пример таквог алгорита је *DES* (енгл. *Data Encryption Standard*) алгоритам за енкрипцију података, чијом се адекватном заменом сматра алгоритам *AES* (енгл. *Advanced Encryption Standard*).

В.3. СЛАБА ХЕШ ФУНКЦИЈА (ЕНГЛ. *WEAK HASHING*)

Употреба слабе хеш функције је претња која је слична претходно поменутој претњи употребе слабог криптографског алгорита, само што се она односи на употребу хеш функција које нису безбедне. Пример такве хеш функције је *MD2* (енгл. *Message Digest Algorithm 2*), док је пример безбедне хеш функције *SHA-256* (енгл. *Secure Hash Algorithm 2*).

В.4. LDAP ИНЈЕКЦИЈА (ЕНГЛ. *LDAP INJECTION*)

Апликације које конструишу LDAP упите за захтеве динамичких веб страница на основу улазних параметара пружених од стране корисника се могу злоупотребити са сигурносног аспекта. Као и код других типова инјекција, исправна валидација достављених улазних података је неопходна како би се спречио напад. Са друге стране, LDAP не садржи пре-компајлиране упите као SQL, услед чега је једини вид заштите добра валидација улазних података.

В.5. МАНИПУЛАЦИЈА ПУТАЊАМА (ЕНГЛ. *PATH TRAVERSAL*)

Манипулација путањама се односи на напад којим се приступа фајловима, директоријумима или командама ван веб кореног директоријума. Безбедност се нарушава тиме што се може доћи до читања фајлова са осетљивим садржајем на веб серверу. Она се извршава обрадом променљивих које указују на фајлове са секвенцама тачка-тачка-коса црта

(„ .. / “) [3]. У случају да на путању фајла може да утиче садржај који уноси корисник, ова претња угрожава безбедност апликације.

В.6. КОЛАЧИЋ БЕЗ ПОСТАВЉЕНОГ АТРИБУТА ЗА СИГУРНОСТ (ЕНГЛ. *SECURE COOKIE FLAG*)

Ова претња се односи на недостатак постављеног атрибута за сигурност (енгл. *Secure*) за колачић који садржи осетљиве податке у оквиру *HTTPS* (енгл. *Hyper Text Transfer Protocol Secure*) сесије. Последица непостављеног атрибута је слање оригиналног садржаја колачића путем *HTTP* сесије, чиме се омогућава увид у садржај колачића од стране трећег лица. Ако се овај атрибут постави, веб претраживач ће спречити слање колачића преко незаштићеног комуникационог канала [3]. У програмском језику *Java* се препоручује коришћење методе *javax.servlet.http.Cookie.setSecure*.

В.7. КРШЕЊЕ ГРАНИЦА ПОВЕРЕЊА (ЕНГЛ. *TRUST BOUNDARY VIOLATION*)

Успостављене границе поверења у програму су повезане са валидацијом коришћених података. Ако су подаци адекватно валидирани, тиме постају подаци од поверења и прелазе поменуту границу. Пример је пријем *HTTP* захтева са корисничким именом који се поставља за чување у сесији без претходне провере аутентикације корисника. Ова претња се избегава коришћењем адекватне валидације и енковања над подацима пре него што се пређе „граница поверења“.

В.8. СЛАБ ГЕНЕРАТОР ПСЕУДОСЛУЧАЈНИХ ВРЕДНОСТИ (ЕНГЛ. *WEAK RANDOMNESS*)

Употреба слабог генератора псеудослучајних вредности је претња која доводи до генерисања вредности које се могу предвидети од стране нападача. Генератори псеудослучајних бројева генеришу секвенце вредности које нису апсолутно случајне, али се могу генерисати секвенце које су теже за предвиђање. На основу тога се разликују статистички и криптографски генератори псеудослучајних бројева, где је излаз статистичких генератора предвидив и може се једноставно репродуковати [3]. Пример генератора у програмском језику *Java* који није безбедан је *Math.random()*, уместо којег се препоручује коришћење *SecureRandom* класе.

В.9. ХРАТН ИНЈЕКЦИЈА (ЕНГЛ. *XPATH INJECTION*)

Попут других типова инјекција, ова претња се манифестује приликом коришћења улазних података унетих од стране корисника за креирање ХРАТН упита. Уколико улазни подаци нису коректно обрађени, нападач може да добије увид у структуру XML података. Критичнија ситуација је ако се користи XML запис за аутентикацију, чиме нападач може да промени своје привилегије [3]. Препорука је коректно валидирају пружени улазни подаци.

В.10. XSS - ИНЈЕКЦИЈА КЛИЈЕНТСКЕ СКРИПТЕ (ЕНГЛ. *CROSS-SITE SCRIPTING*)

Попут других типова инјекција, ова претња се манифестује приликом коришћења улазних података унетих од стране корисника које могу довести до извршавања злонамерних скрипти. Садржај који се шаље веб претраживачу је у већини случајева у формату *JavaScript* кода, али може садржати и HTML код, односно било који код који претраживач може да изврши. На овај начин, нападач може да има увид у осетљиве податке или да уметне код у

оквиру апликације (нпр. лажна форма за пријављивање). Како би се овај напад избегао, препорука је да се користи адекватна валидација улазних података и на серверској и на клијентској страни, која може укључити филтрирање тј. елиминацију одређених карактера попут тагова („<>“), амперсанда („&“) и процента („%“).

В.11. РАЗБИЈАЊЕ HTTP ОДГОВОРА НА ДЕЛОВЕ (ЕНГЛ. *HTTP RESPONSE SPLITTING*)

Ова претња се односи на достављање злонамерних података апликацији, која их затим укључује у оквиру заглавља HTTP одговора. Примарни узрок томе је дозвола обраде података који садрже карактере за позиционирање на леву маргину (енгл. *carriage return*) и пребацивање у нови ред (енгл. *line feed*). Међутим, уколико је платформа која се користи безбедна на унос ових карактера, овај напад није могућ, што је сређено у многим Јава апликационим серверима [3]. Услед тога, ова претња није укључена у оквиру тестова бенчмарка.

В.12. ПРОМЕНА ПАРАМЕТАРА (ЕНГЛ. *PARAMETER TAMPERING*) И ПРОМЕНА ВЕБ-АДРЕСЕ (ЕНГЛ. *URL TAMPERING*)

Промена параметара се односи на промену одређених параметара у оквиру веб-адресе или вредности поља на веб страницама. Прва поменута промена представља претњу промене веб-адресе, која се може посматрати као подскуп претње промене параметара. Узрок овом нападу је недостатак валидације унетих података од стране нападача кроз улазне тачке апликације, где се као пример овог напада може издвојити измена изгледа веб-адресе која садржи параметар броја банковног рачуна. На овај начин, може се потенцијално извршити увид у банковни рачун неког корисника. Овај напад се може избећи адекватном валидацијом улазних података на страни клијента и сервера, која може да укључи дозвољене формате садржаја.