Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/26BE27Majstorovic

FUZZ ТЕСТИРАЊЕ DJANGO REST API-JA

FUZZ TESTING OF DJANGO REST API

Никола Мајсторовић, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У овом раду презентовано је fuzz тестирање Django REST API-ја над апликацијом која представља друштвени веб сајт за међусобну комуникацију и дељење садржаја. У оквиру овог рада описана је целокупна архитектура система за тестирање Ђанго апликације, коју чини алат OWASP ZAP - подржава Fuzz тестирање и Elastic stack (Filebeat, ElasticSearch и Kibana) - колекција три различита пројекта намењена за анализу резултата тестирања. Детаљно је описано конфигурисање свих алата, платформе и радног оквира неопходног за fuzz тестирање предметне Django апликације. Након тога, извршена су fuzz тестирања Django апликације. Анализирани су резултати тестирања.*

Кључне речи: *логовање догађаја, тестирање система, отклањање рањивости*

Abstract – *This paper presents fuzz testing of the Django REST API on an application that represents a social website for mutual communication and content sharing. This paper describes the entire architecture of the Django application testing system, which consists of the OWASP ZAP tool - supports Fuzz testing and the Elastic stack (Filebeat, ElasticSearch and Kibana) - a collection of three different projects intended for the analysis of test results. The configuration of all the tools, platform and framework necessary for fuzz testing the Django application is described in detail. After that, fuzz tests of the Django application were performed. The test results were analyzed.*

Keywords: *Event logging, system testing, vulnerability remediation*

1. УВОД

Тестирање софтверског система представља процес провере и евалуације предметног софтверског система како би се осигурало да комплетан предметни систем задовољи дефинисане захтеве, функционалности, перформансе, сигурности, скалабилности, компатибилности и отпорности на грешке. Могу се примењивати различите технике и методе, укључујући ручно тестирање, аутоматизовано тестирање, симулације, испитивање оптерећења и стрес тестирање [1].

Тестирање представља интегрални део процеса развоја софтвера. Поред исправног функционисања софтвера неопходно је обезбедити и одређени степен сигурности корисника и система који софтверски производ користе. Отклањање сигурносних пропуста у софтверу путем тестирања и у току развоја софтвера у том случају постаје императив. Кључни део заштите информационог система је спречавање недозвољене елевације привилегија, односно заобилажење система контроле приступа.

Једна од метода за проналажење сигурносних пропуста је fuzz тестирање [2].

Предмет овог рада је примена fuzz тестирања софтвера – Django апликације, при чему треба истаћи да се овај метод тестирања користи за откривање грешака, рањивости и непредвиђених понашања у софтверским апликацијама базираним на REST API протоколу. Ова техника се често користи за побољшање сигурности и стабилности софтвера [3].

2. FUZZ ТЕСТИРАЊЕ

Сврха fuzz тестирања је прослеђивање података помоћу аутоматизованих или полу-аутоматизованих техника и тестирање система на различите изузетке. Услед високог степена аутоматизације, овај вид тестирања не захтева много експертског времена као ни претходно познавање софтвера [4].

Састоји од генерисања тест случајева улазних података и праћења извршења тестираног софтвера у току обраде.

Постоји више различитих приступа fuzz тестирању, а најзначајнија су: fuzz тестирање базирано на мутацији и fuzz тестирање засновано на генерисању. Кораци при fuzz тестирању приказани су на слици 2.1 [2].

Слика 2.1. Кораци при fuzz тестирању [2]

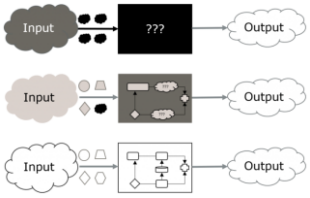
2.1. Методе тестирања софтвера

Метода тестирања софтвера обухвата стратегију и планирање тестирања целог софтверског производа. Метода тестирања укључује одабир одговарајућих техника тестирања за различите делове софтвера, постављање приоритета тестирања, идентификацију ресурса потребних за тестирање и дефинисање циљева тестирања [5].

При тестирању, систем за тестирање (енг. System Under Test, SUT) може да се посматра као црна, бела и сива кутија [6] као што је приказано на слици 2.2.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Горан Сладић, ред. проф.

- Стратегија беле кутије, унутрашња структура система за тестирање је позната.
- Стратегија црне кутије, систем се посматра као црна кутија и не зна се ништа о унутрашњој структури система или коду.
- Стратегија сиве кутије је техника за тестирање апликације са ограниченим познавањем интерног рада апликације и знањем о основним аспектима система.



Слика 2.2 Fuzz тестирање методом црне, беле и сиве кутије

2.2. Технике тестирања софтвера

Техника тестирања софтвера се односи на специфичне методе или приступе које користимо за извођење тестирања софтвера [5].

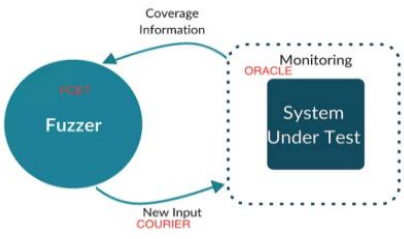
Постоје две основне технике које се примењују код fuzz тестирања. То су dumb fuzz и smart fuzz техника [7].

1. Dumb fuzz техника подразумева приступ код ког се потпуно насумично измењени подаци уносе у апликацију и посматра се реакција апликације на овако насумично измењене податке.
2. Smart fuzz технике уместо креирања потпуно насумичних улазних података, ради се промена специфичних вредности и на тај начин се утиче на основни формат и/или очекивано понашање.

2.3. Начин функционисања fuzz тестирања

Fuzz тестирање се састоји из три кључне компоненте као што је описано на слици 2.3 [8]:

1. "Poet" који креира деформисане улазе или тест случајеве заснованог на генерисању или мутацији у оквиру одређеног протокола, у зависности од тога да ли су улази генерисани од нуле или је потребно модификовање постојећих улаза
2. "Courier" испоручује тест случајеве циљном софтверу. Он може бити: Network protocol fuzzing, File fuzzing, API fuzzing, User interface fuzzing
3. "Oracle" непрекидно врши праћење рада система као и непрекидну процену стања система током одређеног временског периода у односу на његове активности и одређује да ли је тест прошао или није.



Слика 2.3. Опис рада fuzzing-a

2.4. Животни циклус fuzz тестирања

Животни циклус fuzz тестирања се може поделити на следеће кораке:

1. Одабир дела система над којим се врши тестирање
2. Одабир улаза
3. Генерисање fuzz података
4. Извршење тестова коришћењем fuzz података
5. Анализа понашања система
6. Евидентирање проблема

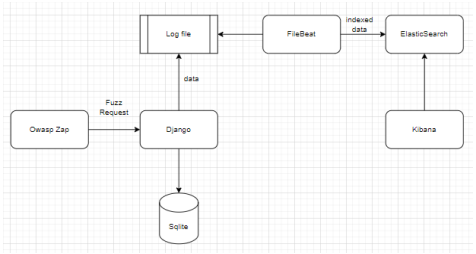
Кораци у fuzz тестирању биће кроз примере објашњени у поглављу 5.5 –Примена fuzz тестирања

3. МОДЕЛ СИСТЕМА

Предметни систем треба да омогући корисницима тестирање Django REST API-ja, као и визуелни приказ резултата, ради увида у понашање поменутог API-ja.

Архитектура система као што је приказано на слици 3.1. састоји се од:

1. OWASP ZAP - алат који подржава fuzz тестирање
2. Filebeat – алат за доставу различитих врста података из лог фајла у ELK (ElasticSearch, Logstash, Kibana) stack ради анализе
3. ElasticSearch – дистрибуирани *multitenant search engine* за претрагу текстуалног садржаја заснован на *Lucene* библиотеци
4. Kibana је софтвер за визуализацију података за *Elasticsearch*
5. Django - веб апликација која је развијена у Django радном окружењу
6. SQlite – увезана библиотека која представља систем за управљање базама података који је уграђен у апликацију



Слика 3.1. Архитектура система

На слици 3.1. описан је концепт fuzz тестирања. Након сваког захтева Django логује одговор у фајл.

Filebeat ослушкује промене у фајлу, прикупља податке које затим структуира и прослеђује у ElasticSearch који их индексира.

Kibana користи ElasticSearch индексе како би омогућио брз приступ и визуализацију података.

4. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

У овом поглављу описани су сви алати, платформа и радни оквир неопходни за fuzz тестирање Django апликације.

4.1. OWASP ZAP

OWASP ZAP (скраћено *Zed Attack Proxy*) је open-source алат за сигурност веб апликација који је развијен од стране *OWASP*-а (*Open Web Application Security Project*). Алат се користи за тестирање

сигурности веб апликација и проналажење рањивости како би се побољшала њихова отпорност на нападе. OWASP ZAP је алат отвореног кода намењен да се користи у циљу тестирања безбедности апликација.

4.2. Filebeat

Filebeat је један од многих лаких „достављача података“ који су доступни као део Elastic Stack-а познатог као Beats. Beats пошиљаоци података имају једну намену и дизајнирани су да буду инсталирани на машини која генерише податке без да имају било какав утицај на перформансе машине. Filebeat чита логове засноване на тексту и прослеђује их било директно у ElasticSearch или у Logstash [10].

4.3. ElasticSearch

ElasticSearch је дистрибуирана платформа за претраживање и анализу података. ElasticSearch је изграђен на пројекту Apache Lucene, који пружа моћне могућности за индексирање и претраживање великих количина података у реалном времену [11].

4.4. Kibana

Kibana је алат за управљање, истраживање, визуализацију и приказивање податке у реалном времену, развијен од компаније која је такође створила ElasticSearch, Logstash и Filebeat (ELK stack) чији је концепт приказан на слици 4.4.

Kibana укључује широк избор додатака и алата који омогућавају корисницима да претражују, врше преглед и анализу индексираних података. Kibana омогућава корисницима да интерактивно истражују, анализирају и визуализирају податке сачуване у ElasticSearch-у. Овај моћан алат се често користи уз ElasticSearch како би се добио увид у податке и идентификовале релевантне информације [12].

4.5. Django framework

Django је популарни open-source веб радни оквир (*framework*) написан у Python који олакшава развој веб апликација. Развијен је од стране *Django Software Foundation*-а и има богат скуп алата и функционалности који омогућују програмерима брзо стварање скалабилних, сигурних и сложених веб апликација [13].

4.6. Sqlite

SQlite је софтверски пакет јавног домена који обезбеђује систем за управљање базом података или RDBMS. Системи релационих база података се користе за складиштење кориснички дефинисаних записа у великим табелама. Поред складиштења и управљања подацима, систем за управљање базом података може да обрађује сложене команде упита које комбинују податке из више табела за генерисање извештаја и података [14].

5. ИМПЛЕМЕНТАЦИЈА СИСТЕМА

У овом поглављу биће приказани и описани сви делови система неопходни за fuzz тестирање Django апликације, као и конфигурација апликације, алата и *engine*-а.

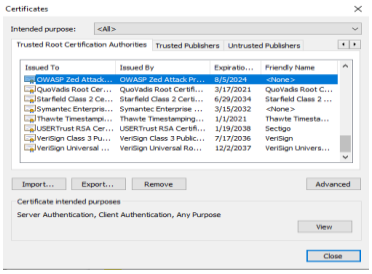
5.1. Конфигурација Owasp Zap алата

Да би се применило fuzz тестирање са OWASP ZAP алатом, потребно је инсталирати OWASP на систем.

Треба нагласити да ZAP захтева Java 11+ да би могао да се покрене.

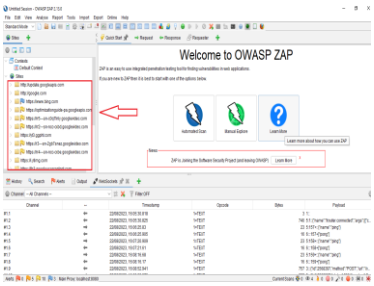
Пошто OWASP не приказује саобраћај локалног домаћина подразумевано у својој историји, потребно је подесити Chrome, или прегледач који ће се користити за тестирање, да користи ZAP proxy, тако што се у proxy подешавању у прегледачу упише локална IP адреса и порт који користи OWASP.

Након поставке, да би OWASP могао да пресреће HTTPS захтеве, потребно је изгенерисати сертификат од стране OWASP-а који увозимо у претраживач, као што је приказано на слици 5.1.



Слика 5.1. Сертификат потписан од OWASP ZAP-а

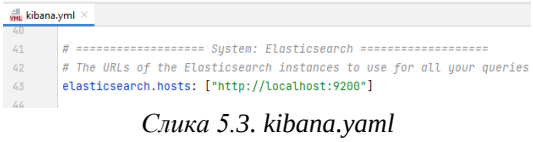
Након поставки и покретања прегледача, у OWASP-у биће приказана историја саобраћаја, као што је приказано на слици 5.2.



Слика 5.2. Историја саобраћаја

5.2. Конфигурација Kibane

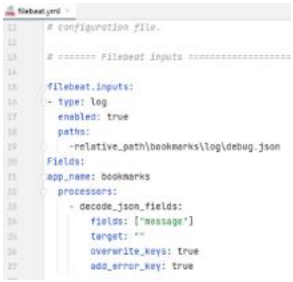
Да би Kibanu интегрисали са ElasticSearch-ом, потребно је изменити kibana.yaml фајл који се налази у директоријуму config у Kibana директоријуму који је потребно преузети, и откоментарисати линију 43, као што је приказано на слици 5.3.



Слика 5.3. kibana.yaml

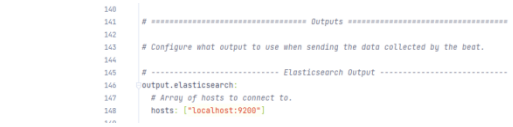
5.3. Конфигурација Filebeat-а

Да би Filebeat могао да региструје промене које се дешавају у Django-вом лог фајлу потребно је додати следеће линије у filebeat.yaml фајл, као што је приказано на слици 5.4.



Слика 5.4. filebeat.yaml

Као и линије кода, које су приказане на слици 5.5.



Слика 5.5. filebeat.yaml

Додавањем линије изнад, Filebeat добија информацију на који излаз шаље трансформисане податке.

5.4. Конфигурација Django лог фајла

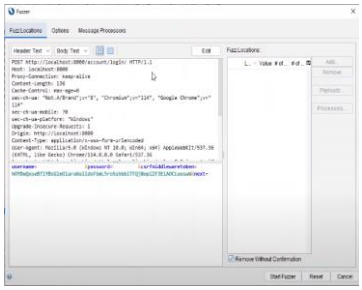
Да би Filebeat могао да прати логове у апликацији, лог фајл мора бити правилно структуиран у JSON формату у ком Filebeat може да га испарсира.

Поља која су значајна за анализирање резултата у Kibani или лог фајлу, приказана су на листингу 5.1. као што је: *agent.id* - означава који је корисник послао захтев; *http.request.method* – означава тип захтева; *http.response.status_code* - означава да ли је одговор успешан; *url.path* представља *endpoint* за приказани одговор; *duration* представља дужину трајања захтева и може да указује на потенцијална уска грла (b*ottleneck*), итд.

5.5. Примена fuzz тестирања

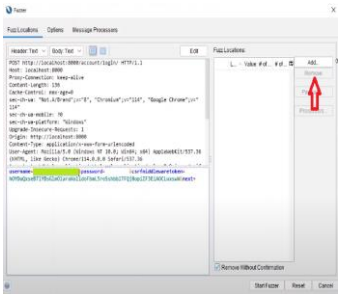
Да би тестирали API, потребно је да OWASP пресретне захтев помоћу прегледача и да на левом прозору буде приказана историја саобраћаја као што је приказано на слици 5.2.

Одабиром одређеног *endpoint*-а и бирањем функције *attack/Fuzz*, отвара се нови прозор који приказује детаље изабраног API-ја, као што је приказано на слици 5.6.



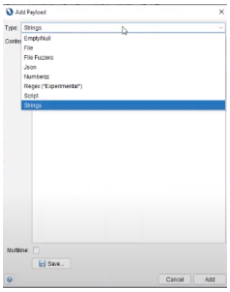
Слика 5.6. Детаљи API-ја који је OWASP снимио

Селектовање одређеног улаза, као што је приказано на слици 5.7, могуће је изменити податке и убацити нове тест случајеве.



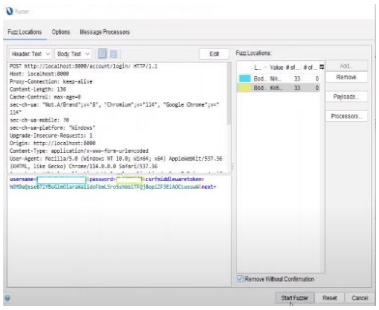
Слика 5.7. Селектовање улаза за генерисање нових тест случајева

Кликом на дугме *Add*, што је означено црвеном стрелицом на слици 5.7, отвара се нови прозор, приказан на слици 5.8, који служи за бирање начина за генерисање тест случајева.



Слика 5.8. Одабир начина генерисања улаза

Након одабира свих улаза и одређивање начина за генерисање тест случајева, као што је приказано на слици 5.9, fuzz тестирање се покреће кликом на дугме *Start Fuzzer*.



Слика 5.9. Одабир изабраних улаза за убацивање тест случајева за тестирање логовања

Ова процедура се понавља за тестирање било ког API-ја,а у наставку ће бити приказан само одабир изабраних улаза за убацивање тест случајева, као на слици 5.8 и анализа резултата тестирања.

5.5.1. Тестирање Login Api-ja

За тестирање *Login* API-ја предметне апликације, потребно је да се у улазне параметре *username* и *password* убаце тест случајеви као што је приказано у тачки 5.5. на слици 5.9.

Као улазни параметри промењиве *username* и *password* изабрани су тест случајеви: *SQL injection* и *Buffer Overflows.*

Након тестирања *Login* API-ја, приказани су резултати.

На графикону број одговора по времену, у интервалу од 15 секунди број одговора је од 2 до 18.

Дужина одговора по времену, дужина одговора не прелази 1 секунд, што имплицира да је одзив апликације задовољавајући.

Свих 2300 одговора, остао је непромењен током периода тестирања (200), што је жељени одговор.

Статус код по времену, остао је непромењен током периода тестирања (200), што је жељени одговор.

Дужина одговара по времену у одређеном интервалу, садржи више од 35000 карактера. Накнадном провером лог фајла, закључено је да је то из разлога, што је велики број карактера убачен у улазни параметар.

На основу приказаних резултата тестирања, закључено је да није дошло до грешке рада API-ја.

Исти поступак се понавља за тестирање логовања администраторске апликације, с тим да је endpoint за администраторску апликацију:
http://localhost:8000/admin/login/

Као и у примеру - тестирање *Login*-а над предметном апликацијом, разлог је велики број карактера убачен у улазни параметар.

5.5.2. Тестирање API-ја за upload слике

Основни предуслов да би се извршило тестирање *upload*-а слике, је да корисник буде улогован у апликацију (да поседује валидан *CSRF token*). За тестирање API-ја за upload слике, потребно је да се у улазне параметре URL-а слике убаце тест случајеви. Као улазни параметри променљиве URL изабрани су тест случајеви: *URI SCHEMAS, XSS XML injection, SQL injection* и *Buffer Overflows.*

Тестирањем API-ја за upload слике, добијени су резултати.

На графикону, број одговора по времену у интервалу од 5 секунди је од 10 до 49.

Дужина одговора по времену је испод 0.012 секунди.

Број одговора, остао је непромењен током периода тестирања (200), што је жељени одговор.

Статус код по времену, остао је непромењен током периода тестирања (200), што је жељени одговор.

Дужина одговара по времену у одређеном интервалу је уједначена и садржи око 2400 карактера по захтеву.

На основу приказаних резултата тестирања, закључак је да није дошло до грешке рада овог API-ја.

5.5.3. Тестирање API-ја за брисање изабраног корисника из апликације

Претпоставка да би се извршило тестирање брисања корисника из апликације је да корисник који тестира буде улогован у апликацију (да поседује валидан *CSRF token*) као администратор.

Након тестирања API-ја за брисање изабраног корисника из апликације и посматрања података у бази података, закључено је да су подаци остали не-промењени, односно да корисник који нема улогу ад-министратора не може да обрише другог корисника.

6. ЗАКЉУЧАК

У овом раду посебан нагласак стављен је на примену fuzz тестирања користећи OWASP ZAP алат, као и Elastic stack (Filebeat, ElasticSearch и Kibana) за визуализацију и анализу резултата тестирања, како би се идентификовале потенцијалне рањивости у API–јима у Django апликацији која је тестирана.

Интеграција OWASP ZAP алата са Elastic stack-ом пружа могућност ефикасног прикупљања резултата тестирања и анализу истих, чиме се добија дубљи увид у потенцијалне рањивости API-ја, као и степен озбиљности рањивости. У раду је детаљно описано конфигурисање OWASP алата, Elastic stack-а и Django апликације за праћење логова.

Детаљно су тестирана четири API-ја користећи OWASP ZAP алат за генерисање и слање различитих облика неважећих и неочекиваних података.

Након пажљиве анализе резултата користећи Kibanu као алат за визуелизацију података, може се закључити да није идентификована ниједна критична или озбиљна рањивост у анализираним API-јима.

Резултати изложени у раду показују да су имплементиране безбедносне мере у тестираним случајевима на одговарајућем нивоу и да су API-ји отпорни на познате облике напада који су симулирани током fuzz тестирања.

7. ЛИТЕРАТУРА

[1] “Automated Software Testing. 1999”, Dustin, E.

[2] “Detekcija sigurnosnih propusta faz testiranjem”, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2013, Nikolić Aleksandar, Sladić Goran, Milosavljević Branko, Konjović Zora.

[3] “REST API Fuzzing by Coverage Level Guided Blackbox Testing”, Chung-Hsuan Tsai Shi-Chun Tsai; Shih-Kun Huang, 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)

[4] ”Fuzz testiranje REST API-ja”, Univerzitet u Novom Sadu, Fakultet tehničkih nauka 2021, Tamara Veličković

[5] “Metode I tehnike testiranja softvera” 2021, Sveučilište u Zagrebu Ekonomski fakultet Zagreb, Informatički menadžment, Andreja Mamić

[6] “Neke metode za automatizovano testiranje softvera”, Univerzitet u Beogradu Matematički fakultet, Kruna Matijevic

[7] “Primena Fuzz testiranja na DMS softver po SDL metodologiji” Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2010, Kristina Stojaković

[8] What is Fuzzing, © 2017 Synopsys, Inc, https://www.synopsys.com/content/dam/synopsys/sig-assets/whitepapers/what-is-fuzzing.pdf

[9] “Study of the techniques used by OWASP ZAP for analysis of vulnerabilities in web applications”, 2022, Linköping University | Department of Computer and Information Science, Adam Jakobsson

[10]”SCADA STATISTICS MONITORING USING THE Elastic Stack (Elasticsearch, Logstash, Kibana)”, 2017, James Hamilton, Brad Schofield, Manuel Gonzalez Berges, Jean-Charles Tournier CERN, Geneva, Switzerland

[11] ”Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and analytics engine”, 2015, Clinton Gormley, Zachary Tong

[12] “Kibana 7 Quick Start Guide: Visualize your Elasticsearch data with ease”, 2019, Anurag Srivastava

[13] “Beginning Django Web Application Development and Deployment with Python”, 2017, Daniel Rubio

[14] “Using Sqlite”, 2010, Jay A. Kreibich

Kratka biografija:

Никола Мајсторовић рођен је у Новом Саду 1993. године. Мастер рад на Факултету техничких наука из области Рачунарство и аутоматика – Електронско пословање одбранио је 2023. године контакт: majsta93@hotmail.com