ACADEMIA TEHNICĂ MILITARĂ

**Facultatea de Sisteme Informatice și Securitate Cibernetică**

Specializarea: Calculatoare și sisteme informatice pentru

apărare și securitate națională



**Proiect de diplomă**

**IDENTIFICAREA VULNERABILITĂȚILOR SOFTWARE FOLOSIND TEHNICI DE FUZZING**

Coordonator științific:

Col. prof. dr. ing. Ion Bica

Absolvent:

St. sg. maj. Marius RADU

Conține \_\_\_\_\_\_\_\_\_\_\_ file

Inventariat sub nr. \_\_\_\_\_\_

Poziția din indicator: \_\_\_\_

Termen de păstrare: \_\_\_\_\_

**BUCUREȘTI**

**2020**

Pagina alba

Pagina alba

Pagina alba

Pagina alba

Pagina alba

**Abstract:**

Vulnerabilities consists in programming errors or configuration errors of software applications, if not identified and correct in time, can be exploited for a potential attacker. Therefore, they can be controlled by security, which can have control over the aces and have access to the confidential date or can block service functions. Vulnerable software can result in a cybersecurity and care issue facing the organization at the moment.

Many of the software applications currently being developed at the organizational level are based on Web technologies. In addition, applications displayed on the Internet are also web applications. Therefore, the necessary vulnerabilities in terms of application advice are identified and corrected, it is critical to create before the application of those applications in that application.

Fuzzing is a testing technique for a software application, such as black box, care found in malformed / semi-malformed data injections in an automatic way in order to identify potential vulnerabilities. The attack vectors in case of fuzz testing are the inputs / outputs of the software application. In the case of web applications, they are represented by URLs, forms, user-generated content, etc.

This project aims to identify the methods and tools used for fuzz testing and to develop a guide on the application of these techniques to identify vulnerabilities in Web applications. The last subject of this project is about illustrating and implenting a fuzzer to discover vulnerable web applications to cross-directory (directory traversal) attacks.

Keywords: exploit cybersecurity, vulnerable, black box, malformed, attack vectors, fuzzer, URL,

**Rezumat:**

Vulnerabilitățile reprezintă erori de programare sau greşeli de configurare ale aplicațiilor software care, dacă nu sunt identificate și corectate la timp, pot fi exploatate de către un eventual atacator. Prin intermediul acestor breșe de securitate, atacatorul poate obține controlul asupra sistemului respectiv, poate avea acces la date confidențiale sau poate bloca funcționarea serviciilor. Vulnerabilitățile software reprezintă principala cauză a problemelor de securitate cibernetică cu care organizațiile se confruntă în momentul de față.

O mare parte dintre aplicațiile software dezvoltate în momentul de față la nivel organizațional sunt bazate pe tehnologii Web. În plus, aplicațiile expuse în Internet sunt, de asemenea, aplicații Web. Prin urmare, identificarea și corectarea vulnerabilităților specifice acestui tip de aplicații, este critic să se facă înainte de darea în exploatare a aplicațiilor respective.

Fuzzing-ul este o tehnică de testare a aplicațiilor software, de tip black box, care constă în injecții de date malformate / semi-malformate într-un mod automat cu scopul de a identifica potențiale vulnerabilități. Vectorii de atac în cazul fuzz testing-ului sunt intrările / ieșirile aplicației software. În cazul aplicațiilor Web, acestea sunt reprezentate de URL-uri, formulare, conținut generat de utilizator, etc.

Acest proiect are ca obiective identificarea metodelor și uneltelor folosite pentru fuzz testing și elaborarea unui ghid privind aplicarea acestor tehnici pentru identificarea vulnerabilităților în cadrul aplicațiilor Web. De asemenea, în ultima parte a documentului, se va prezenta un fuzzer propriu care să permită descoperirea aplicațiilor Web vulnerabile la atacuri de tip directory traversal.

Cuvinte cheie: securitate cibernetică, vulnerabil, black box, malformat, vectori de atac, fuzzer, URL.

Cuprins:

[1. Introducere 11](#_Toc11413593)

[1.1 Importanța temei alese 12](#_Toc11413594)

[1.2 Scopul ṣi obiectivele lucrării 12](#_Toc11413595)

[1.2 Metodologia de cercetare 13](#_Toc11413595)

[1.2 Rezultatele obţinute 14](#_Toc11413595)

[1.2 Rezumatul lucrării pe capitole 15](#_Toc11413595)

[2. Vulnerabilităţi software 16](#_Toc11413596)

[2.1 Tipuri de vulnerabilități 16](#_Toc11413597)

2[.2 Metode de detecție a vulnerabilităților **.**](#_Toc11413598)17

[2.3 Vulnerabilități specifice aplicațiilor Web 20](#_Toc11413598)

[2.4 OWASP Top 10 21](#_Toc11413598)

[3. Fuzz testing 24](#_Toc11413599)

[3.1 Tehnici de fuzz testing 24](#_Toc11413600)

[3.2 Componentele unui fuzzer 25](#_Toc11413601)

[3.3 Clasificarea fuzzer-elor 25](#_Toc11413602)

[3.4 Avantajele și limitările unui fuzzer 27](#_Toc11413603)

[3.5 Exemple de fuzzere pentru aplicații Web 28](#_Toc11413604)

[3.6 testarea aplicațiilor Web folosind tehnici de fuzzing 30](#_Toc11413605)

[4. Descrierea implementării 32](#_Toc11413607)

[4.1 Definirea cerinţelor 32](#_Toc11413608)

[4.2 Prezentarea arhitecturii soluţiei 33](#_Toc11413609)

[4.3 descrierea modulelor componente 34](#_Toc11413610)

[4.4 tehnologii folosite pentru implementare 40](#_Toc11413611)

[4.5 rezultatele testelor efectuate și interpretarea acestora 41](#_Toc11413612)

[4.5.1 Compararea rezultatelor cu Burp-Suite Intruder 45](#_Toc11413612)

[5. Concluzii 48](#_Toc11413620)

5[.1 Rezumatul constatarilor 48](#_Toc11413621)

[5.2 Directii viitoare de cercetare ..49](#_Toc11413622)

[6. BIBLIOGRAFIE 50](#_Toc11413623)

Listă de figuri

Fig. 1 Structura generală a unui fuzzer 27

Fig. 2.0 ZAP funcționând ca man in the middle proxy 28

Fig. 2.1 ZAP conectată direct la un proxy 29

Fig. 3.0 Burp Suite-setarea target-ului 31

Fig. 3.1 Burp Suite- setarea payload-urilor 32

Fig. 4.0 Diagrama de clase al proiectului practic 34

Fig. 4.1 Pagina principală al proiectului 36

Fig. 4.2 Opțiuni valabile input-urilor testate 36

Fig. 4.3 Fereastră pentru setarea cookie-urilor 37

Fig. 4.4 Fereastră pentru setarea autentificării 38

Fig. 4.5 Fereastră pentru adăugare alte setări http 38

Fig. 4.6 Funcționalitatea decoder al proiectului 39

Fig. 4.7 Funcționalitatea comparer al proiectului 39

Fig. 4.8 Funcționalitatea requests al proiectului 40

Fig. 4.9 Rezultatele obținute de fuzzer 42

Fig. 4.10.0 Informații generate de fuzzer potrivit testelor anterioare 44

Fig. 4.10.1 Informații generate de fuzzer potrivit testelor anterioare 44

Fig. 4.11 Implementarea aplicației vulnerabile DVWA 44

Fig. 4.12 Rezultate obținute de fuzzer folosind căutarea in adâncime 45

Fig. 4.13.0 Interceptarea datelor folosind Burp Suite 46

Fig. 4.13.1 Setarea payload-urilor folosind Burp Suite 46

Fig. 4.14.0 Rezultatele obținute de Burp Suite 47

Fig 4.14.1 Rezultatele obținute de fuzzer-ul propriu 47

Listă de abrevieri

`

# Introducere

Tema aleasă, **„Identificarea vulnerabilităților** **software folosind tehnici** **de fuzzing”,** reprezintă un subiect important în contextul securității cibernetice. Tehnicile de fuzzing sunt critice în cazul testării unei aplicații pentru a identifica și eventual soluționa diferite erori de programare înainte ca aplicația respectivă să se lanseze, ținând cont de faptul că dezvoltarea și numărul aplicațiilor crește semnificativ. Fuzzerele pot să testeze diferite tipuri de aplicații, în principiu, cele în care serverul primește input de la utilizator. și în urma unor teste efectuate, pe baza unor input-uri malformate, aceste fuzzere pot salva aplicațiile și firmei o avere.

Majoritatea, dacă nu, toate aplicațiile în care se dorește securitatea la un nivel înalt, tehnici de fuzzing au fost efectuate și pentru fiecare perioadă de mentenanță, update, aceste tehnici se pot repeta, menținând mereu o securitate stabilă. Cele mai comune tipuri de erori sunt cele de “buff overflow” în care introduce un șir de date mai lung decât lungimea oferită de server, și astfel se produc erori și un comportament nefiresc al aplicației atacate.

Datorită numărului mare de aplicații care apar, dezvoltarea acestora și implementarea securității acestora s-a realizat într-un mod prea rapid, astfel încât sa asigure aplicațiilor securitatea și o rată de apariție a bugg-urilor foarte scăzută. De aceea tehnicile de fuzzing joacă un rol important în testarea aplicațiilor in intervalul de creare/dezvoltare și cel de lansare, identificând problemele de securitate și îmbunatățind flow-ul aplicației.

De preferat este ca aplicaţiile să utilizeze tehnicile de fuzzing încă din perioada de dezvoltare, pentru a reduce costurile şi a îmbunătăţi securitatea aplicaţiei, şi de a salva timp depanatorilor, încă de la început. În cadrul acestei teme ne vom focusa pe tehnicile de fuzzing în contextul aplicaţiilor Web, care pot prezenta multe vulnerabilităţi. Cum aplicaţiile Web prezintă o mare influenţă a economiei globale, putem să spunem că tehnicile de fuzzing joacă un rol important în cadrul acestui context şi redau semnificativ importanţa temei alese.

Un alt aspect important al tehnicilor de fuzzing este dat de costul redus. Desigur că nicio aplicaţie nu este perfectă, şi niciun fuzzer nu poate identifica toate vulnerabilităţile şi bugg-urile unei aplicații dar fuzzerele pot să ofere rezultate satisfăcătoare, în raport cu timpul alocat. Însă este un domeniu unde este necesară continuarea cercetării, pentru a ține pasul cu dezvoltările generale ce se petrec în domeniul Web.

## Scopul ṣi obiectivele lucrării:

Scopul lucrării este de a analiza modalitatea în care un fuzzer poate identifica vulnerabilitățile existente în cadrul aplicațiilor web, de a vorbi despre cele mai relevante fuzzere web și descrierea acestora după diferite criterii. De asemenea un obiectiv important al lucrării este de a prezenta un fuzzer propriu care identifică vulnerabilitățile de tipul “Directory Traversal” tot în cadrul aplicațiilor web.

Obiectivul principal este de a prezenta atât rolul fuzzer-elor în contextul actual, de a analiza modul în care acestea pot pătrunde în aplicații, și evident, modul în care ne putem proteja aplicațiile de aceste tehnici de fuzzing. După care vom analiza un fuzzer de tipul “Directory Traversal” și vom discuta rezultatele acestea pe un server web dedicat cercetării vulnerabilităților aplicațiilor Web.

## Metodologia de cercetare

Metodologia de cercetare s-a realizat în cadrul a două mari părți, am putea spune, complementare.

În prima parte facem o introducere a vulnerabilităților, a metodelor de detecție ale acestora. Vorbim despre vulnerabilități în cadrul aplicațiilor web, după care analizăm principalele vulnerabiliăți emise de o organizație competentă. Ulterior, după ce ne-am familiarizat cu vulnerabilitățile, vorbim despre tehnicile de fuzzing care urmăresc să exploateze “greșeli” de programare. Facem o trecere în revistă a componentelor şi a clasificării acestora, după care, folosind un fuzzer relevant, vom vedea cum arată testarea unei aplicaţii web folosind acele tehnici de fuzzing pe care fuzzer‑ul le conține.

În cea de-a doua parte, vom vedea un exemplu de fuzzer de tipul Directory Traversal, pornind de la arhitectura acestuia, la descrierea modulelor si tehnologiilor folosite, vom discuta despre rezultatele și conlcuziile unor serii de teste aplicate unei aplicații web vulnerabile, cu scop educativ, care ajută doritorii să-și perfectioneze abilitățile de fuzzing în cadrul aplicațiilor web.

## Rezultatele obținute

În urma realizării testelor pentru identificarea vulnerabilităților, se pot trage o serie de concluzii după care ne putem ghida în vederea realizării unui nivel de securitate cât mai ridicat. Acestea fac referire atât la modul de operare al fiecărui fuzzer în mod particular, cât și la importanța realizării testelor cu ajutorul lor.

Se poate sesiza faptul că utilizarea tehnicilor de fuzz testing, folosind o arhitectură potrivită, pot obține o serie de atacuri de mărimi mari într-un interval destul de scurt. Rezultatele obținute de fiecare fuzzer în parte au fost clasificate cu ajutorul unor categorii care permit o mai bună vizualizare a modului de lucru și a utilizării acestuia în diferite contexte.

S-a putut identifica faptul că aplicațiile web pot fi securizate cu ajutorul fuzzerelor. Ținând cont că tehnicile de fuzzing sunt de tip black box și implicit automatizate, putem rula în paralel aceste tehnici, pentru o mai bună comparație a unor scenarii diferite din punct de vedere al arhitecturii/codului țintă.

Datorită numeroaselor funcționalități, al nivelului ușor de utilizare dat de interfața grafică și al vitezei de putem spune că Burp Suite Intruder reprezintă un fuzzer optim care poate fi utilizat în vederea identificării vulnerabilităților. Un alt fuzzer care excelează in acest domeniu este ZAP (The OWASP Zed Attack Proxy, care are de asemenea un fuzzer încorporat pe care îl putem utiliza. Spre deosebire de Burp Suite, acesta nu este accelerat în timp și toate funcționalitățile pe care le oferă sunt gratuite. Este evident că niciun fuzzer nu poate identifica toate vulnerabilitățile unei aplicații, dar ce este recomandat este să utilizăm cât mai multe fuzzere care să facă teste în paralel pentru a optimiza securitatea aplicației.

Concluzia care a putut fi trasă în urma implementării practice este aceea că tehnica de fuzzing de tipul “Directory traversal” este o tehnică ce poate aduce informații imporante atacatorului, ce merită sa fie cercetată și cunoscută de orice dezvoltator.

## Rezumatul lucrării pe capitol

Primul capitol al lucrării realizează o scurtă prezentare a importanței temei lucrării. Tot în cadrul acestui capitol sunt prezentate atât scopul și obiectivele disertației, cât și metodologia de cercetare aleasă. De asemenea conține o serie de concluzii asupra rezultatelor obține în cadrul implementării practice.

În cadrul celui de-al doilea capitol au fost studiate cele mai importante vulnerabilități care se regăsesc la nivelul aplicațiilor web și modul de detecție ale acestora. După clasificarea acestora și a modului de detecţie prin care putem evita asemenea vulnerabilităţi, vom discuta despre cele mai relevante Top 10 vulnerabilităti prezentate de OWASP (The Open Web Application Security Project). Aceasta este o comunitate online care produce articole, metodologii, documentații, instrumente și tehnologii disponibile în mod liber în domeniul securității aplicațiilor web.

În cadrul celui de-al treilea capitol vom discuta despre fuzz testing. Vom identifica tipurile de fuzz testing, componentele acestuia, clasificarea lor, și vom face o analiză a avantajelor și dezavantajelor acestora. După care, vom vorbi despre câteva din cele mai cunoscute și utilizate fuzzere Web la momentul de față. Incheierea capitolului va fi dat de descrierea etapelor de testare a unei aplicații web folosind tehnicile de fuzzing.

În cadrul celui de-al patrulea captiol vom analiza un fuzzer personal de tipul “Directory Traversal”, unde vom prezenta arhitectura proiectului, tehnologiile si modulele folosite și in cele din urmă vom realiza niște teste pentru care vom face o sinteză și o comparație a rezultatelor, cu scopul de a identifica modul în care ne putem proteja și putem exploata aceasta metoda particulară a fuzzing-ului, prin care, vom vedea, încercăm să accesăm date din serverul web țintă, care ar trebui să nu poată fi accesate de utilizator.

Al cincilea capitol este cel care definește concluziile proiectului, precum sinteaz principalelor idei din lucrarea de licență, cât și niste sugestii către continuarea cercetării.

Ultimul capitol, cu numărul șase, reprezintă bibliografia utilizată în elaborarea lucrării actuale.

# Vulnerabilitățile aplicațiilor

## Tipurile de vulnerabilități

În funcție de utilitățile unei aplicații, de arhitectura, și bineînteles complexitatea acesteia, pot apărea, în diferite etape ale ciclului de viață, în mod involuntar diferite tipuri de vulnerabilități de care dezvoltatorul să nu fie conștient. Astfel, poate apărea în scenă atacatorul care poate să extragă diferite informații, dacă a găsit vulnerabilitatea respectivă pe care ulterior să o exploateze.

În ultimii ani, numărul aplicațiilor a crescut exponențial. Multe firme lansează aplicații într-un interval de timp scurt, scăzând timpul de testare și verificare a aplicației semnificativ. Această viteză influenţează în mod direct securitatea aplicaţiilor, slăbind-o și oferind atacatorilor oportunități de infilrtrare în soft-ul vulnerabil.

Depinde de situații, aceste erori de programare și/sau dezvoltare se pot rezolva mai repede sau mai târziu, la fel cum și costul pierderilor generate, pot fi mici sau mari, în funcție de perioada în care aplicația a fost în mentenanță, sau în funcție de datele care au fost descoperite de atacator.

Conform listei CWE / SANS Top 25, există trei tipuri principale de vulnerabilități de securitate:

### Apărare defectuoasă

Apărarea defectuasă se referă la măsuri de apărare implementate greșit, care nu reușesc să protejeze aplicația împotriva posibilor atacuri. Există diferite tehnici de apărare, inclusiv autorizarea, criptarea și autentificarea. Atunci când sunt folosite corect, aceste tehnici au capacitatea de a proteja organizația înlăturând o mare parte din posibilile atacuri cibernetice. Însă, dacă implementarea lor este slabă, creează o iluzie de securitate, în timp ce programul software este predispus la diferite atacuri.

### Gestiunea slabă a resurselor

Practicile de gestionare a resurselor includ transferul, utilizarea, crearea și chiar distrugerea resurselor din cadrul unui sistem. Atunci când gestionarea resurselor este slabă sau nu oferă siguranță deplină, aplicația în caz dvs are tendința de a avea vulnerabilități precum Directory Traversal (despre care vom vedea o implementare practică in capitolul numărul cinci) , Buffer Overflow, și multe altele.

### Conexiune instabilă între elemente

Atunci când interacțiunea dintre componentele sistemului și / sau rețelei, este nesigură, aplicația este expusă la multe amenințări, inclusiv SQL injection, open redirect, cross-site scripting și multe altele.

Pentru a se asigura lipsa unor astfel de vulnerabilități, trebuie să se acorde cea mai mare atenție modului în care datele circulă prin rețelele și sistemele utilizate. Dacă se poate asigura circulația datelor, majoritatea vulnerabilităților și amenințărilor menționate mai sus sunt soluționate. Trebuie totuși să se acorde atenție fiecărui tip de vulnerabilitate în parte și să se ofere soluții relevante pentru acestea.

O altă clasificare a tipurilor de vulnerabilități , impărțite în trei clase, conform [1]:

* Vulnerabilităţi datorate etapei de design
* Vulnerabilităţi datorate etapi de implementare
* Vulnerabilităţi operaţionale.

## 2.2 Metode de detectie ale vulnerabilitatilor

Utilizarea unui model și a unei arhitecturi recomandate alături de revizuirea aplicației și testarea aplicației în diferite etape ale ciclului de viață sunt utile pentru a preveni și a detecta posibile vulnerabilități, însă, este necesar ca securitatea acesteia să se bazeze și pe niște instrumente care pot fi utilizate de programatori pentru a detecta vulnerabilitățile în timpul procesului de construcție a software-ului. Unele dintre aceste instrumente se bazează pe metode statice, astfel încât nu este necesară rularea codului sursă pentru a testa prezența anumitor vulenrabilități.

În cazul metodelor dinamice, codul este rulat într-un mediu dedicat pentru a efectua testarea și colectarea bucăților de cod care pot produce aceste vulnerabilități. În secțiunea următoare vă prezentăm unele static și dinamic tehnici de detectare a vulnerabilităților.[4]

## 2.2.1 Tehnici statice

Tehnicile statice sunt cele accesează în mod direct codul sursă fără a rula aplicația, obiectivul fiind de a evalua sau de a obține informații specifice într-un mod pasiv. Există diferite tehnici pentru a efectua testarea statică. Mai jos apar câteva exemple relevante:[2]

* **Pattern Matching**

Consistă în căutarea unui șablon „pattern” în codul sursă și oferă ca rezultat

numărul de apariții ale acestuia. De exemplu, dacă luăm în considerare limbajul C, modelul ar putea fie orice apel către funcții periculoase (vulnerabile) cum ar fi „getc”, care citește un caracter de la tastatură. Aeastă metodă de căutare poate fi ușor implementată, folosind un regex, însă este limitată, deoarece depinde de forma exactă a acelui șir de caractere, și poate genera un număr fals pozitiv de apariții, dat fiind faptul că nu există nicio analiză a rezultatelor.

* **Analiza lexicală**

Analiza lexicală adaugă un pas suplimentar înainte de aplicarea unui Pattern Matching. De fapt, codul sursă este transformat într-o secvență de token-uri, care sunt comparate ulterior cu o bază de date vulnerabilă pentru a le identifica. Numărul fals pozitiv este de asemenea ridicat pentru că nu ia în considerare sintaxa sau gramatica programului.

* **Parsing**

Parsing-ul este mai complex decât analiza lexicală, deoarece reprezentarea programului este construită folosind un arbore de parsing, pentru a analiza ulterior sintaxa și semantica programului. De exemplu, tehnica de parsing este des utilzată pentru a detecta atacurile de injecție SQL (SQL injection).

* **Calificatori de tip (Type Qualifiers)**

Calificatorii de tip sunt folosiți pentru a califica/clasifica tipurile de date și pentru a modifica proprietățile variabilelor din limbajele de programare. De exemplu, const int este un calificator de tip reprezentat de o constantă întreagă,in timp ce  int este corespondentulde tip necalificat, fiind un simplu integer. Cqual este folosit pentru a specifica și a verifica proprietățile programelor C folosind califactori de tip definiți de utilizator, care sunt adăugați ulterior la program. Programul modificat este analizat pentru a găsi vulnerabilități.

* **Analiza fluxului de date**

Scopul este de a determina valorile posibile pe care le poate avea o variabilă sau o expresie pe parcursul executării programului, adecvat special pentru detectarea reîncărcărilor tampon. Autorii iau reguli care descriu tiparele de vulnerabilitate și codul sursă pentru a detecta locațiile și căile modelului în program. Procesul este executat în trei părți: potrivirea modelului, controlul și analizor de flux și de date.

* **Analiza nuanțelor**

Este un caz special de analiză a fluxului de date, în cazul în care orice date provenite de la încredere surse, de ex. introdus de un utilizator, este o problemă potențială în sistem, astfel este marcat ca îmbrăcat. Fluxul de date neclintit este monitorizat, deoarece nu poate atinge critici funcționează cu excepția cazului în care este procesat și schimbat în netratat.

* **Verificarea modelului**

Verificarea modelului este o tehnică de testare automată care poate fi utilizată dacă modelul sistemului îi îndeplinește specificațiile și ulterior pentru a detecta vulnerabilitățile. De obicei, verificarea modelului este a tehnică complexă, pentru că elaborarea modelului este dificilă(se bazeaza pe recunoașterea unui FSM (final state model)), însă, odată obținută, devinefoarte ușor de testat proprietățile respectivului soft.

**2.2.2 Tehnici dinamic**

Tehnica dinamică constă in rularea codului sursă, în a analiza răspunsul aplicației la input-urile testate, și ulterior în a trage niște concluzii despre vulnerabilitatea aplicației în acel context.

* **Injectarea defectuoasă(Fault injection)**

Injectarea defectuasă este o tehnică de testare care introduce în sistem date cu caracter defectuos/malițios pentru a testa comportamentul sistemului. De menționat este faptul că trebuie să cunoasțem anumite date pentru a obține niște eventuale atacuri, spre exemplu limbajul de programare în care a fost scris programul, sistemul de operare pe care rulează (în cadrul aplicațiilor web), etc. După testarea intrărilor malițioase, recomandat este focusarea pe zona unde s-au obținut rezultate diferite, și să se continue cercetarea în acea direcție.

* **Fuzz Testing**

Ideea acestui tip de test este de a furniza date aleatorii ca date de intrare în aplicație pentru a determina dacă aplicația le poate gestiona în totalitate corect. Testarea fuzzing este mai ușor de implementat decât fault injection, deoarece nu este necesară cunoașterea a multor informații din sistem pentru efectuarea respectivelor teste. De asemenea, dimensiune unui fuzz pot fi imense, iar cu cât timpul de testare crește, se pot descoperi vulnerabilități noi. După ce o vulnerabilitate a fost găsită, un fuzzer, un instrument în cadrul fuzz testing-ului, poate fi folosit pentru a identifica cauzele apariții acelei vulnerabilități.

* **Dynamic Taint**

“Vopseaua dinamica” rulează în cadrul unui program şi observă care computaţii sunt afectate de source taint predefinite, cum ar fi input-ul generat de utilizator.[3]

* **Data Sanitization**

Data sanitization(iginiezarea datelor) resrezintă metoda prin care se face ștergerea permnaentă, din memoria device-ului respectiv.După execuția acestui proces, device-ul nu mai prezintă date reziduale, acestea nu pot fi recuperate nici folosind un tool special care să caute în memorie aceste date. Această metodă poate avea un caz special în care un dispozitiv hardware se conectează la dispozitivul țintă și distruge in mod fizic datele de pe acesta.

## Vulnerabilități specifice aplicatiilor Web

Vulnerabilitățile web nu au fost un subiect important la începuturile tehnologiei web, întrucât nu existau date care ar fi tentat atacatorii în a exploata diferite aplicații web. Însă, cu trecerea timpului, securitatea site-urilor web a crescut, s-au diminuat numărul de bugguri, se folosesc arhitecturi mai avansate, securizate, însă, direct proporțional au apărut și din ce în ce mai multe date confidențiale, conturi bancare, date cu caracter personal care circulă prin internet, iar atacatorii folosesc diferite tipuri de atacuri bazându-se pe diferite vulnerabilități specifice. Un motiv pentru care vulnerabilitățile încă prezintă o mare problemă este dată de modul rapid în care apar aplicațiile web. Cererea de aplicații este mare, iar timpul dedicat pentru securitate este mic, ceea ce poate crea vulnerabilități și erori de programare. În funcție de confidențialitatea și importanța datelor stocate, dezvoltatorii încearcă să-și imbunătățească securitatea verificând daca o anumită vulnerabilitate este prezentă, după care, să reducă cât de mult posibil gradul de risc pe care îl generează. Aceste vulnerabilități diferă în timp, dar în subcapitolul următor vom descrie principalele top 10 vulnerabilități, conform OWASP, din contextul actual.

## 2.4 OWASP TOP 10

The Open Web Application Security Project®  (OWASP) este o organizație non profit care lucrează in a îmbunatăți securitatea aplicațiilor, cu proiecte software open source, sute de sedii locale în tot globul, zeci de mii de membrii, este sursa principală oentru dezvoltatori și tehnologi în cadrul securității web.

Top 10 OWASP este un document standard de informare pentru dezvoltatori cu privire la securitatea aplicațiilor web. Conține informații și sinteze cu privire la cele mai importante riscuri de securitate pentru aplicațiile web. Este cunoscut la nivel internațional de dezvoltatori ca fiind cel mai solid standard pentru acest domeniu.[4]

Orice dezvoltator, care se ocupă de o aplicație in cadrul tehnologiei web, trebuie să fie la curent cu cele mai improtante riscuri de securitate și vulnerabilități din contextul actual. Mai jos vedem câteva exemple.

Top 10 Riscuri de securitate pentru aplicații web conform OWASP:[5]

* **Injection (Injecție de date).** Această metodă este cea mai întâlnită, și constă in trimiterea de date invalide către server în speranța obținterii unui comportament neobișnuit al serverului țintă, ulterior obținând informații neautorizate utilizatorului fără drepturi speciale. Cele mai comune atacuri de tip injecție, sunt SQL injection, în care trimitem query-uri care ar putea conține date relevante, sau al ștergerii anumitor date prin intermediul paginii url, sau al oricărui tip de input. Reușita provine din implementarea minimală al validării și al permisiunilor serverului. Evitarea acestui tip de atac se poate face prin folosirea unui Api sigur, care interpetează query-urile, limitarea query-urilor trimițând utilizatorului datele strict necesare, folosirea unui “whitelist” care permite executrea unor query-uri care apar în acea listă, eliminarea caracterelor speciale din cadrul query-ului primit pentru a evita orice tentativă de fuzz testing asupra acelui query.
* **Broken Authentification( Autentificare coruptă)** Implementarea deficitară al funcțiilor de autentificare pot permite diferite atacuri asupra serverului, precum atacuri automate folosind dicționare de date asupra credentialelor, brute force, forțarea opțiunilor de ‘forgot password’, ‘recovery', intrebările cunoscute de utilizator la crearea contului, etc. Lipsa experienței dezvoltatorilor, cât și graba lansării aplicației fac pozibilă prezența acestor tipuri de vulnerabilități. Metodele prin care ne putem proteja aplicația de asemenea tip de atac este prin implementarea autentificării multi-factor, verificarea ca parola să aibă diferite pattern-uri, asigurarea că metodele legate de parolă, cum ar fi recuperarea acestuia, schimbarea parolei folosesc un API securizat. De asemenea, limitarea și crearea unui delay dupa fiecare încercare de autentificare reduce semnificativ posibilitatea de atacuri brute force sau dicționar. Folosind un manager de tip server-side putem asigna ID-uri random după fiecare logare, de asemenea, id-ul sesiunii nu ar trebui să fie inclus in url-ul paginii, ci stocat într-un mod securizat și invalid după termianrea sesiunii.
* **Sensitive Data Exposure (Expunerea datelor sensibile).** Aplicațiile de Banking, Medicale și cele PII (Personal indentifiable information) sunt doar niște exemple în care datele trebuie să fie securizate într-un mod avansat. În caz contrat, atacatorii pot fura sau modifica astfel de date slab protejate pentru a efectua fraude cu cardul de credit, furt de identitate sau alte infracțiuni. Aceste tipuri de date trebuie menținute în mod criptat, trasnmiterea acestora trebuie să se transmite folosind un PKI care să mențină un grad ridicat de securitate. Metode de protejare ar ar fi clasfiicarea datelor procesate, stocate, sau transmise de fiecare aplicație, evitarea stocării datelor care nu sunt necesare, folosirea agloritmilor eficienți, și un management bun al cheilor , etc.
* **Entități externe XML (XXE).**Aceste atacuri sunt eficiente mpotriva aplicațiilor care primesc ca input date sub forma unui XMl, sau care parseză acele XML-uri. Enintiățile XML porst configurate evaluează referințele entităților externe din documentele XML. Entitățile externe pot fi utilizate pentru a dezvălui fișiere interne folosind gestionarea URI a fișierelor, partajarea fișierelor interne, scanarea porturilor interne, atacurile de refuz de serviciu și executarea codului de la distanță. Majoritatea parserelor XML sunt, by default, vulnerabile la atacuri de tip XEE, de aceea implementare și crearea unui mediu securizat este responsabilitatea principala a dezvoltatorului. Ne putem proteja de aceste tipuri de date folosind un set mai viabil de stocare al datelor, cum ar fi JSON, evitarea serializării datelor, updatarea și menținerea la curent al procesoarelor și librăriilor folosite de aplicație, implementare unui whitelist de tip server side,care să verifice input-urile primite, blocarea entităților externe XML și procesarea DTT în cazul procesării parsării XML în aplicație, etc.
* **Broken Acces Control(Controlul corupt de acces).** Restricțiile privind ceea ce utilizatorii autentificați au voie să facă nu sunt adesea aplicate în mod corespunzător. Atacatorii pot exploata aceste defecte pentru a accesa funcționalități și / sau date neautorizate, cum ar fi accesul la conturile altor utilizatori, vizualizarea fișierelor sensibile, modificarea datelor altor utilizatori, modificarea drepturilor de acces etc.
* **Configurare greșită de securitate**. Configurarea greșită a securității este vulnerabilitatea care răspunde pentru majoritatea problemelor legate de securitatea aplicațiilor, brute force fiind principalul inamic al acestui tip de vulnerabilitate.Configurăriile incomplete, afişarea mesajelor de eroare care pot conţine informatii sensibile, stocarea în cloud nesecurizat sunt principalele exemple de securitate greşită. Alături de brute force, există diferite atacuri derivate care prezintă succes, cum ar fi: “Defecţiuni neabordate, configuraţii implicite, fişiere şi directoare, servicii inutile”.[6]. Ne putem feri de cele menţionate anterior setând restrictii pentru utilizarea de fişiere şi gestionarea controlului comentariilor, al utilizatorilor şi al vizibilităţii datelor.
* **Cross Site Scripting(XSS).** Defecțiunile de tip XSS apar atunci când atacatorul introduce date maliţioase, mai exact script‑uri într-un website, și folosesc acel site, în mare parte pentru a redirecționa utilizatorii către pagini cu reclame, cu conținut malițios. Ce este de menționat este că aceste vulnerabilități depind de interacțiunea cu utilizatorul, prin accesarea unei pagini sau prin social engineering. O metodă de reducere a riscului asupra atacurilor de tip XSS ar fi folosirea de tehnologii de tip firewall specializate valabile, de ex Sucuri Firewall.
* **Insecure Deserialization (Deserializare nesigură**). Orice obiect este de fapt o structură de date. Transmiterea datelor se face la nivelul bytes-ilor. În consecință apar două procese:serializare și deserializare. Deserializarea nesigură, vulnerabilitate in care se folosec date necunoscute sau suspicioase, poate duce la executarea de la distanță a codului, implicând un atac de tip DoS. Cea mai bună metodă de apărare împotriva acestor tipuri de atacuri este de a nu accepta obiecte serializate din surse suspicioase.

* **Utilizarea componentelor cu vulnerabilități cunoscute**. Dacă unele din modulele externe pe care le folosim sunt predispuse la anumite vulnerabilități , aplicația noastră este de asemenea predispusă. De aceea trebuie să ne alegem ce dependențe externe folosim și să eliminăm cele ce nu sunt necesare în cadrul aplicației.
* **Logare și monitorizare insuficiente**. Înregistrarea și monitorizarea într-un mod ineficient, pot aduce daune în cadrul aplicației web. Dacă un atacator începe să intre în sistem și acest lucru nu este semnalizat printr-un log file, spre exemplu, acesta poate continua să provoace daune mari aplicației web, până vreun dezvoltator să sesizeze acest lucru.[6]

# Fuzz Testing

## Tehnici de fuzz testing

Fuzzing-ul este o “tehnică pentru testarea negativă, iar testarea negativă nu este nimic nou domeniul asigurării calității. Testarea hardware acum câteva decenii deja conținea negativ testarea în mai multe forme. Cea mai tradițională formă de testare negativă în hardware se numește injecția eronată"[8], tehnică de care am discutat capitolul anterior. Fuzzing-ul este o tehnică de testare de tip black box în care sistemul testat primește input-uri de date, într-un mod automatizat, cu scopul de a identifica vulnerabilități, erori, bugguri, etc. Programe și framework-uri specializate sunt folosite pentru crearea de fuzz testing și testarea acestora sunt numite fuzzere. În ultimii 15-20 de ani fuzzing-ul a evoluat foarte mult, dintr-o simplă testare într-un domeniu voluminos al fuzz testing-ului cu suport de dezvoltare, și comunități online. Cea mai cunoscută organizație în acest sens fiind OWASP. Câțiva termeni care descriu testele similare cu procesul de fuzzing sunt următorii:”[8]

* Testare negativă;
* Mutația protocolului (Protocol mutation);
* Testarea robusteții (Robustness testing**). „Testarea robusteții este orice metodologie de asigurare a calității axată pe testarea robustetei software-ului. Testarea robustetei a fost utilizată și pentru a descrie procesul de verificare a robustetei (adică corectitudinea) cazurilor de testare într-un proces de testare”[9].**
* Testarea sintaxei;
* Injecția defectelor;
* Rainy-Day testing;
* Dirty Testing.

În acest capitol vom pune accentul pe modul de funcționare al fuzzerelor și vom face o clasificare a acestora, pe baza căreia vom prezenta dezavantajele și avantajele, după diferite criterii, după care, vom face o scurtă prezentare a unui exemplu de testare porning de la un fuzzer de care am discutat anterior.

## Clasificarea Fuzzer-elor

Există numeroase moduri de a clasifica fuzzerele. Aceste moduri depind de ţintă (sistemul de operare pe care rulează, tipul de aplicaţie, etc.), vectorii de atac utilizaţi, şi multe altele. Însă, una dintre cele mai bune metode de a clasifica fuzzerele este după modul în careinput-urile sunt generate. Conform [10], acestea pot fi:

* **Fuzzeri de mutație( Mutation fuzzers)**

Aceste tipuri de fuzzere crează un rezultat aleatoriu, pentru un input dat, în urma unor operații, și/sau combinații de șiruri de date cu potențial malițios. Un exemplu ar fi adăugrea unui șir de caractere foarte lung asupra inputului pentru a testa dacă aplicația țintă este eroare de tipul “buffer overflow”.

* **Fuzzeri Generatori (Inteligenti)**

Fuzzerii Generatori crează input independent de input-ul existent, în loc să încerce mutații pe acel input. Însă, fuzzer-ul trebuie să cunoască anumite date, un nivel minim de cunostiințe despre protocolul pe care îl folosește aplicația să primească acele date.În comparație cu fuzzerii de mutație, acest tip de mutație va trebui să conțină diferite informații, ca să fie eficient. De menținut este că este mai dificil de creat față de fuzzerii de mutație.

## Componentele unui fuzzer

Fuzzerele moderne nu au ca scop doar testarea vulnerabilității prin trimiterea input-urilor, ci acestea conțin pe langă funcționalități și isntrumente care imbunătățesc atat automatizarea procesului de fuzzing, cât și sintetizarea erorilor/bugg-urilor obținute. Conform [10], avem trei componente principale:

* **Fuzzing baseline**
* **Input Generation** (generarea de input)
* **Application Monitoring** (monitorizarea aplicației)

Inputul care este trimis este generat prin acest “fuzzy baseline”[10].In principal sunt două surse de unde informaţia este extrasă:

* **Traffic sniffing** (este o componentă a fuzzer-ului care, pe baza unui pachet de date, este în măsură să genereze alte input-uri pe care ulterior, să le folosim în procesul de fuzzing.
* **Protocol template(** este asemănător traffic sniffing-ului, doar că acesta generează input-uri noi, care să respecte regulile date de protocolul specificat. Mai este cunoscut sub numele de “**protocol-based fuzzing**”[10].

O altă clasificare, este dată de [11], care împarte în 5 componentele unui fuzzer, care este, părerea mai, mai consistentă, bazându-se mai mult pe structura logică a acestuia:

* **Modelarea protocolului**: Pentru funcționalitatea fuzzer-ului în diferite contexte, acesta trebuie să cunoască protocolul pe care aplicația țintă îl folosesțe, și să creeze un mediu satisfăcător pentru ca fuzzer-ul să poată interacționa cu aceasta
* **Anomaly library(Biblioteca de anomalii)**: Colecțiii de date cu potențial malițios. Acestea trebuie alese în funcție de protocolul serverului țintă și in raport cu direcțiile de cercetare.
* **Attack simulation engine**: Acest modul foloseşte vectori de atac, şi generează un set de date de fuzz testing.
* **Runtime analysis enginge**: Un modul care generează, in paralel cu procesul de fuzzing, oferă date referitoare la procesul curent
* **Reporting**: Salvarea rezultatelor relevante într-un format care să ajute dezvoltatorii în vederea cercetării ulterioare.

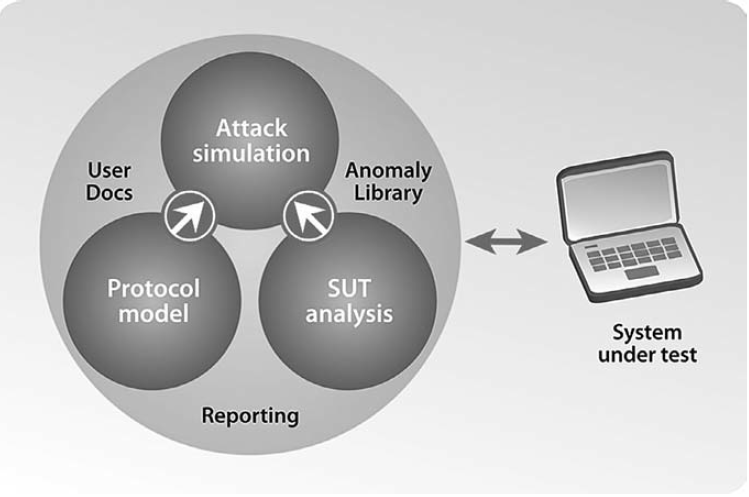


Fig. 1 Structura generală a unui fuzzer

# 3.4 Avantajele și limitările unui fuzzer

“Un avantaj important al fuzz testing-ului este acela că design-ul test-ului este extrem de simplu, și fără preconcepții despre comportamentul sistemului”.[11]. De asemenea, un avantaj al fuzz-testing-ului, ar fi acela că rezultatele pot fi proporționale cu timpul. Cu cât timpul de testare crește, cu atât numărul de bugg-uri/vulnerabilități sunt descoperite. Totdată , fuzzing-ul lucrează la nivel computațional ridicat și indentifică erori pe care, deseori, omul nu le-ar fi putut identifica. În plus, fuzzing-ul este unul dintre puținele metode de a testa un system care este închis în totalitate.

Discutând despre limitările fuzzer‑elor, principala problemă pe care o produce un fuzzer este accea că, în majoritatea cazurilor, găsește doar erori foarte simple. Cu cât fuzzerul cunoaște mai mutle date despre un anumit protocol, cu atât mai puține erori vom găsi. De aceea căutarea exhaustivă/ aleatoare este încă cea mai folosită in contextul fuzzing-ului.

O altă limitare al fuzzing-ului este aceea că, dorind să facem niste blackbox testing, de obicei dispozitivul este oprit, iar interpretarea rezultatelor și al importanței acestora se face intr-un mod dificil, ținând cont că debugg-ul este de asemenea absent.

Totuși, fuzz testing-ul este o ramură importantă a securității web, care oferă rezultate pe care, oamenii, în majoritatea cazurilor nu le-ar fi descoperit, și care nu trebuie ignorată de niciun dezvoltator care își dorește un nivel de securitate ridicat pentru aplicația sa.

# 3.5 Exemple de fuzzere pentru aplicații web

Niciun fuzzer nu poate identifica, precum nicio aplicație nu este fără erori. Însă diferite fuzzere pot obține rezultate mai bune sau mai slabe, în funcție de cât de bine se pliază funcționalitățile fuzzer-ului pe eventualele vulnerabilități ale aplicației țintă. Drept urmare, este indicată verificarea securității prin rularea în paralel a diferitor fuzzere. Vom vorbi inițial despre fuzzer-ul principal al organizației OWASP, despre care am vorbit în capitolul anterior:

**3.5.1 OWASP ZAP ( Zed Attack Proxy)**

Conform [12], este cea mai folosită aplicaţia la nivel global pentru scanarea aplicaţiilor web. Este gratis şi open source, întreţinută activ de un grup internaţional dedicat şi voluntarial. Aplicaţia este cunoscută pentru atacuri de tip “man in the middle proxy”. Aplicaţia stă intre browser şi aplicaţia ţintă, analizează traficul, şi când este cazul, modifică conţinutul pe care ulterior îl trimite server‑ului, pentru a testa anumite vulnerabilități. Poate fi folosită ca și o aplicație de sine stătătoare, sau ca și o aplicație daemon.”Program care rulează ca proces în background, fără să fie controlat de interacțiunea cu utilizatorul” [13].

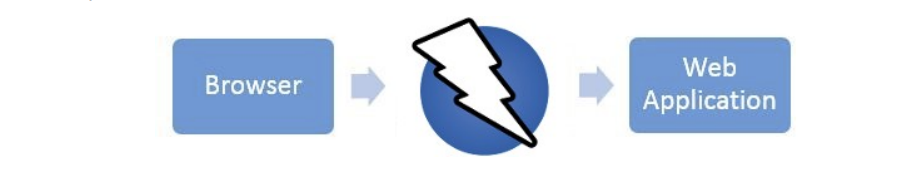


Fig. 2.0 ZAP funcționând ca man in the middle proxy

Dacă exista deja un proxy în rețea, ZAP poate să se coneceteze direct la acel proxy.

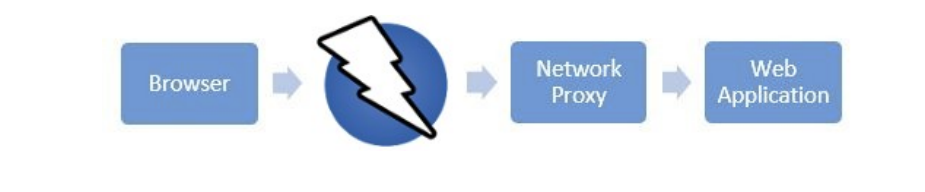


Fig. 2.1 Zed Attack Proxy conectată direct la un proxy

Interfaţa grafică este destul de prietenoasă, uşor de utilizat. Cei de la owasp de asemenea oferă guide-uri şi documente de documentaţie pentru utilizarea aplicaţiei. În continuare o să vorbim despre cele mai importante funcţioanlităţi pe care le prezintă ZAP, conform [14]:

* Interceptarea Proxy –este principalul lucru care trebuie inclus, deoarece ajută la vizualizarea traficului, interceptarea și modificarea acestuia din mers. Multe lucruri se pot întâmpla între interceptarea unui request și trimitea request-ului modificat.
* Scanarea automată – analizează securitate aplicației prin simularea de atacuri a unei aplicații dinamice
* Scanarea pasivă – Această funcționalitate nu atacă direct apicația, însă analizează taficul și răspunsurile serverului
* Scanare Brute Force- aplică brute force pe fișiere și directoare
* Spidering- spidering-ul ajută la construirea structurii unui website, încercând să gasească diferite link-uri în acel website asemănător cu tehnica de fuzzing de tipul directory Traversal)
* Fuzzing
* Port Scanning: incearcă să afle porturile libere ale aplicaţiei web
* Certificate SSL dinamice: ajută la interceptarea de request-uri sau response-uri catre sau de la server.
* BeanShell Console dialog: utilizat pentru scrierea de script-uri in Java
* ZAP este a aplicaţie de nivel internaţional, suportând 11 limbi, excluzând limba engleză.

**3.5.2 Sulley Framework**

Sulley este un framework de fuzz-testing, automat, care constă în multiple componente extensibile. În comparație cu ZAP, acesta nu dispune de o interfață grafică, însă dispune de un server web,de obicei situat în localhost, în care se prezintă statusul procesului de fuzzing curent. Sulley este un framework puternic, având ca scop nu doar simplicarea reprezentării datelor, ci și a monitorizării datelor. Dacă cele mai multe fuzzere se focusează pe generarea datelor, sulley nu numai că generează date, însă urmărește rețeaua și monitorizează starea aplicației țintă, fiind capabil să determine ce secvențe unice au creat greșeli/erori. [15]

**3.5.3 Burp Suite Intruder**

Burp Suite este o un tool ce se ocupă cu automatizarea atacurilor asupra aplicațiilor web. Este foarte puternic și prezintă o largă gamă de funcțioanlități, pornind de la brute force pe directoare până la exploatarea de tipul SQL injection. Aceasta se bazează pe cereri HTTP și le modifică în numerioase moduri, în funcție de analiza și cererile utilizatorului și analizează răspunsul server-ului la aceste request-uri. Ca și mod de comunicare cu utilizatorul, interfața grafică pe care o oferă Burp Suite este de asemenea, prietenoasă, oferă diferite informațiii despre termenii importanți legat de fuzzing. Acestea fiind spuse, poate fi utilizat și de un începător din acest domeniu, fapt pentru care l-am inclus în această listă.

Alte fuzzere care merită atenție sunt următoarele:

Ca framework:

* Peach Fuzzer

Fuzzere Specializate:

* 4JbroFuzzer
* WSFuzzer
* Wfuzz
* Spike Proxy
* TAOF (The Art Of Fuzzing)
* WebFuzz

**3.6 Testarea aplicațiilor Web folosind tehnici de fuzzing**

Atunci când ne dorim să aplicăm conceptul de fuzzing pe o aplicație web, trebuie sa analizăm modul în care clientul comunică cu aplicația. Care sunt datele care sunt primite ca input de aplicație, care sunt parametrii utilizați, ce protocoale folosește, ce headere folosește aplicația, etc.

După ce am identificat aceste noțiuni, va trebui să să selectăm datele care vor fi trimise spre aplicație, mai pe scurt, payload-urile. Internetul conține o mulțime de payload-uri complexe pentru majoritatea vulneraribilităților web, majoritatea dintre ele fiind pe [www.github.com](http://www.github.com). Pe langă aceste payload-uri deja existente, diferite fuzzere au opțiunea de a genera propriile fuzzere în funcție de algoritmii implementați.

Având payload-urile configurate, rămâne doar să le introducem în fuzzer-ul pe care îl folosim, să incepem procesul, și să îl lăsăm să-și facă treaba. Recomandat este să lucrăm cu diferite fuzzere în paralelel pentru a observa rezultatele și a compara eficența fuzzer-elor.

Folosind Burp Suite cel mai simplu mod de a seta pagina este din tab-ul Intruder, submeniul Target.

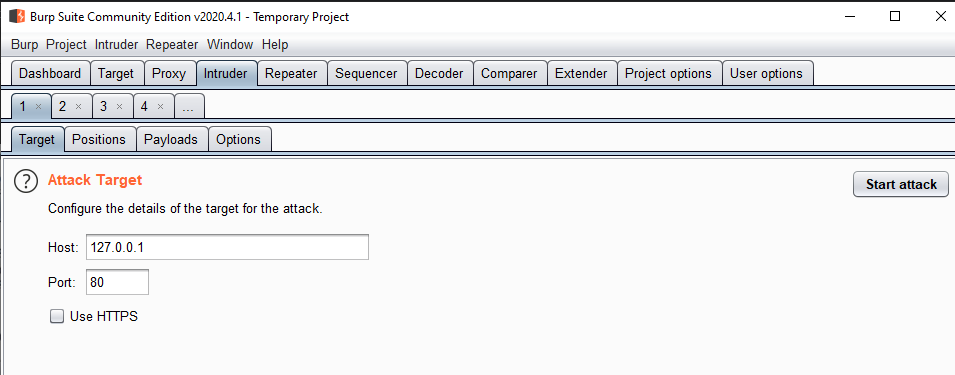


Fig 3.0 Burp Suite, setarea target-ului

De asemenea, paylaod-urile se pot seta din subemniul alăturat, Payloads.

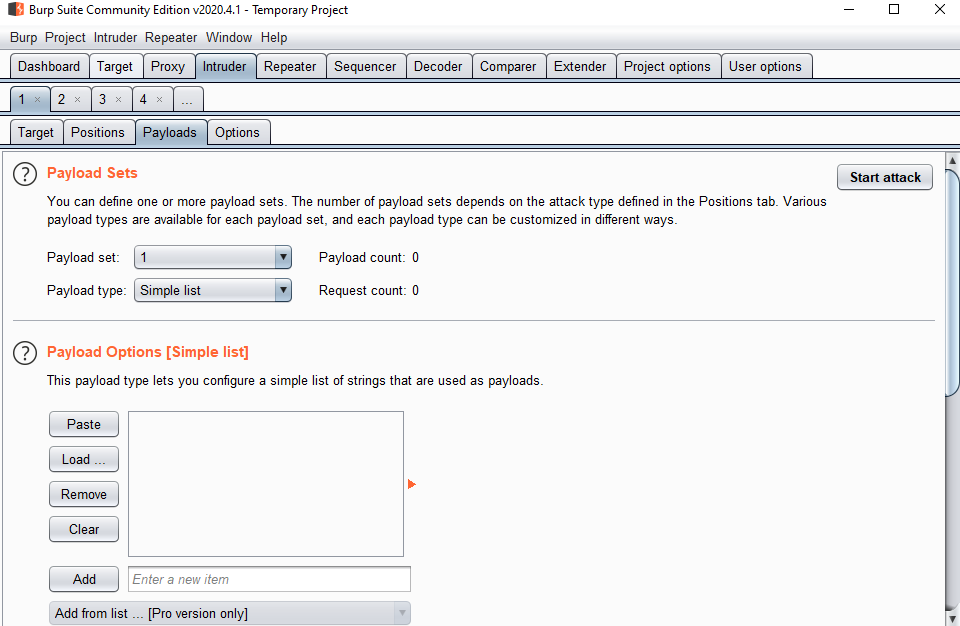


Fig 3.1 Burp Suite, setarea payload-urilor

După terminarea procesului de fuzzing, rămâne de internreptat și analizat rezultatele. În funcție de paylaod-uri și vulnerabilitatea exploatată, trebuie urmărite diferite aspecte, precum cuvinte cheie, lungimea conținutului,etc

# Descrierea implementării

Dacă in capitolele anterioare am discutat despre tipurile de vulnerabilități, tehnicile de fuzzing , avantajele și limiările acestora, în acest capitol se vor descrie arhitectura și funcționalitățile propriului fuzzer destinat atacurilor de tipul “Directory Traversal” asupra applicațiilor Web.

De asemenea, acest capitol urmărește in a prezenta tehnologiile și modulele folosite, transpunerea conceptelor studiate anterioare în cod, analiza rezultatelor obținute in urma unor teste, cât și un ghid de utilizare al aplicației. Toate acestea se vor realiza folosind limbajul de nivel înalt Python. În cadrul proiectului vom folosi latura de orientare de obiecte pe care o oferă Python.

Scopul aplicației create este de a identifica vulnerabilități ale aplicațiilor Web, printr-o interfață ușor de utilizat, utilizând tehnologii concrete perioadei actuale, care oferă informații structurate de analiză a testelor generate și conține tool-uri ce pot eficientiza următoarele cercetări.

## Definirea cerințelor

Principala cerință a unui fuzzer de tip directory traversal, este acela de a genera structurat și eficient request-uri către serverul țintă. Procesul de fuzzing în cazul atacurilor de tip *Directory Traversal in aplicații Web* folosește “brute force” sau o listă de input‑uri pentru a trimite server-ului request-uri cu potențial malițios.

A două cerință importantă este aceea de a permite utilizatorului să definească și să adauge request-ului diferite opțiuni(cookie-uri, fisiere de payload, date de autentificare, etc.) , pentru a crea o sesiune de fuzzing relevantă în raport cu serverul web țintă. Aceste opțiuni se vor modifica/seta printr-o interfață grafică, folosind o librărie specializată din Python. Setările vor fi structurate în funcție de categorii, iar cele mai relevante se vor afla pe pagina principală a aplicației.

A treia cerință ar fi ilustrarea rezultatelor operației de fuzzing într-un mod lizibil, care să permită oricărui utilizator o ulterioară cercetare și niște statistici ale procesului. Acele date vor fi puse într-o pagină separată, cu formă tabelară, care să diferențieze request-urile, în funcție de “reușita” de atac.

O ultimă cerință principală este dată de prezența unor funcționalități care permit continuarea investigației în urma unui test(posibilitatea de a crea request-uri personalitate, de a coda/decoda string-uri, de a compara două texte). Aceste funcționalități pot să salveze timp utilizatorului, în funcție de sarcinile acestuia.

## Prezentarea arhitecturii soluţiei

Pentru realizarea aplicaţiei demonstrative au fost utilizate următoare componente:

* PyCharm(2019 2.5) – un IDE pentru Python (3.8)
* Diverse librării din Python(*Tkinter-*pentru interfața grafică, *requests*-pentru crearea de request-uri, *concurent-futures*-pentru viteza de lucru folosind thread-uri, etc.)

Scopul aplicației demonstrative este de a testa vulnerabilitatea serverelor Web cu ajutorul unui fuzzer ce se bazează pe tehnica “directory traversal”. Acesta, pe baza unor input-uri și ale unor setări date de utilizator, trimite request-uri malițioase cu scopul de a găsi vulnerabilități în serverul web țintă.

Arhitectura aplicației este dată de relația clasei FuzzEngine, care se ocupă cu toate funcționalitățile fuzzer-ului, cu clasa Applicațion, de unde primește datele necesare configurării procesului, și unde trimite rezultatele în timp real. Mai jos voi atașa o diagrama a claselor importante din cadrul proiectului:

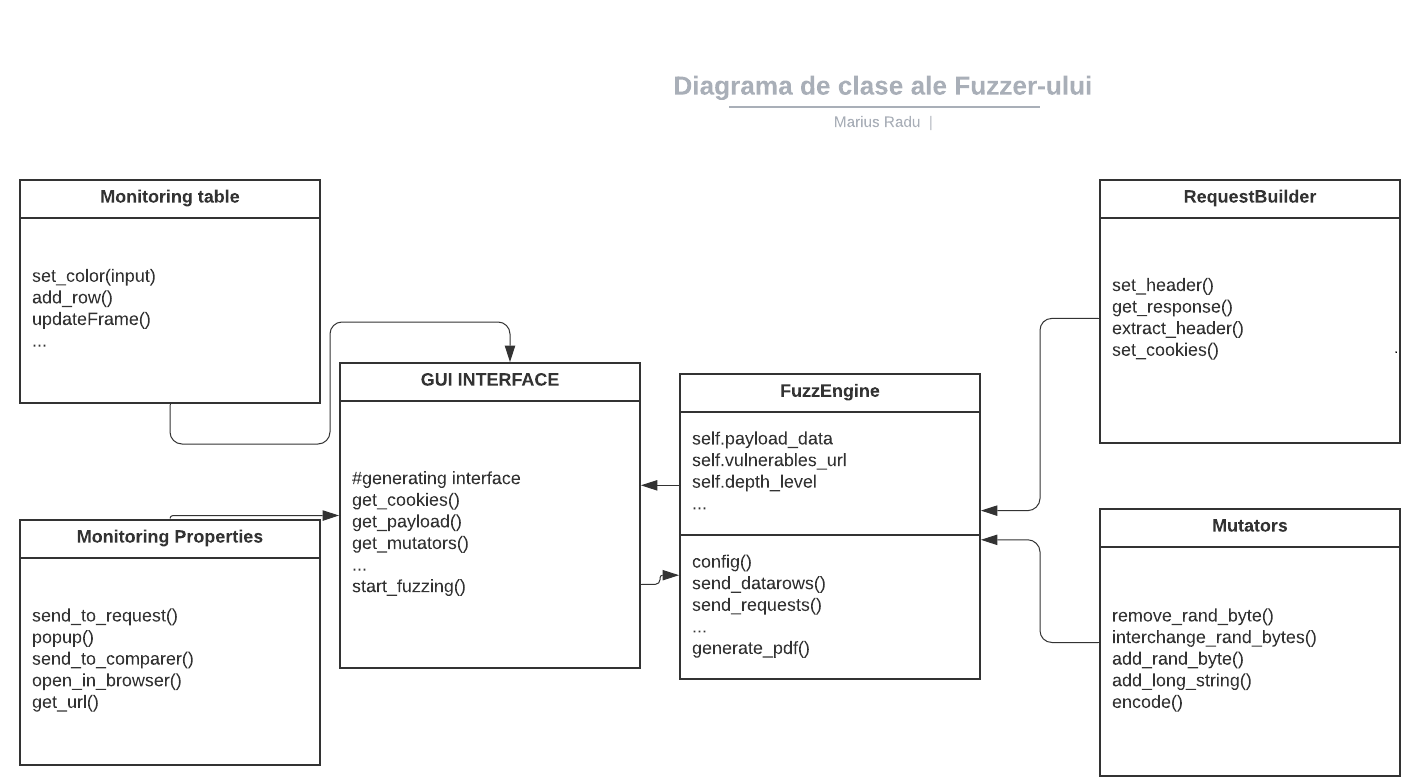


Fig 4.0 Diagrama de clase al proiectului practic

## Descrierea modulelor componente

Componentele de bază din cadrul aplicației sunt date de clasa *FuzzEngine*, *Mutators*, unde se generează input-urile cu potențial malițios, clase interconectate cu clasa principală al interfaței grafice, *Application*. Ne vom concentra pe componenta de de fuzzing(FuzzEngine, Mutators) fiind mai relevante în cadrul temei discutate.

Clasa *FuzzEngine*, îndeplineşte următoarele funcţionalităţi:

* Folosind un obiect de tipul *Session*, din modulul requests, urmăreşte ca request-urile generate de utilizator să nu fie respinse de server adăugând in header-ul acestora: cookie-uri, date de autentificare, server proxy, user-agent, etc, în funcție de datele introduse de utilizator. Aceste funcționalităti sunt utile in situația în care observăm un link care ar putea fi vulnerabil, dar ca să accesăm această pagină trebuie să fim logați și/sau conțin niste cookie-uri specifice. De asemenea se pot genera request-uri personalizate prin intermediul paginii “Request” din submeniul “Tools” al interfeței grafice. Request-urile pot fi de tipul “GET” sau “HEAD”. Depinde de server, “HEAD” poate ține locul lui “GET” pentru rezultate mai rapide, conform OWASP.
* Folosind python pur, în concordanță cu datele introduse de utilizator, generează fluxul de date ce urmează a fi testat, astfel: Pentru fiecare url din fișierul/fișierele de payload, se crează input-uri noi în funcție de nivelul de adâncime de căutare din directorul rădăcină, de metodele de alterare prezente în clasa Mutators, și de posibile poziţii din url unde probabilitatea ca input-urile să pătrundă prin securitatea server-ului web este crescută.
* După ce s-au generat toate input-urile, gestionează și comunică cu interfața grafică statusul fiecărei cereri în parte, statistici ale intregii operații de fuzzing. Aceste statistici sunt trimise de asemenea către metoda *write\_to\_pdf(..),* din cadrul classei FuzzEngine, care folosind librării utile pentru lucrarea cu pdf-uri(*reportlab*) și grafice(*matplotlib*) generează un fișier pdf cu informațiile importante din cadrul ultimei sesiuni.

Clasa *Mutators* este folosită pentru a adăuga, modifica și a de șterge octeți/șiruri de octeți sau string-uri de lungime mare cu potențial malițios.(terminatori de şir, caracterul *null* , etc.)

Clasa Applicațion este folosită pentru afișarea interfaței grafice, și de a stoca datele introduse de utilizator prin aceasta. De asemenea conține un obiect de tipul FuzzEngine care se instanțiază după pornirea fuzzer-ului, sau după salvarea setărilor. Aplicația este impărțită in 4 submeniuri:

* *Basic configuration*: Pagina principală unde setăm majoritatea atributelor importante procesului de fuzzing(Adresa, proxy-ul, numărul de thread-uri, nivelul de adâncime, tipurile de metode utilizate pentru request-uri, etc.)

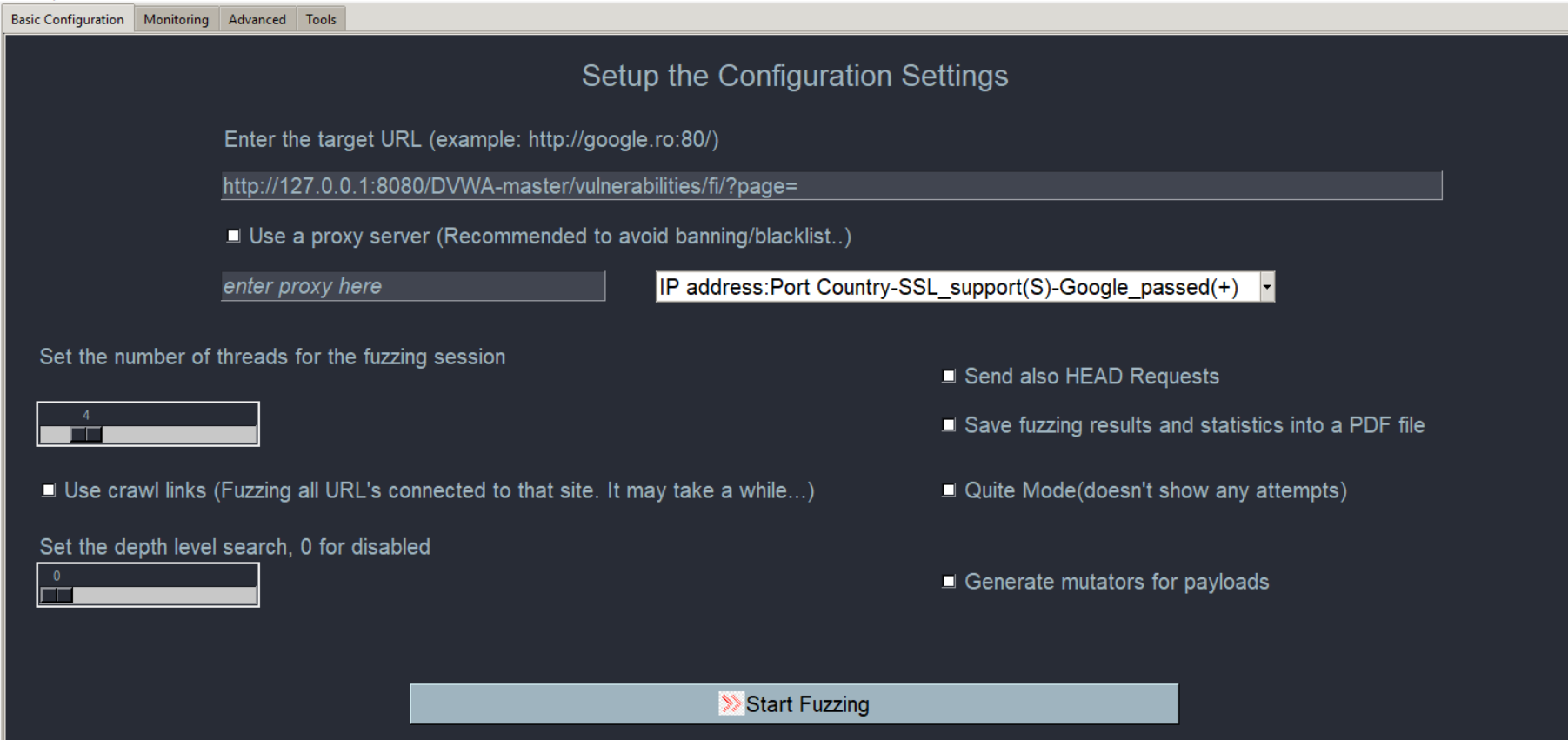


Fig. 4.1 Pagina principală al proiectului

* Monitoring: Fereastră ce conţine informaţii ale procesului de fuzzing în timp real. Request-urile cu rezultate pozitive după căutare, request-urile cu sta­tusul (200, 301(Moved), 302(Found), 400( Unauthorized, 403(Internal Erro­r), 500(Internal Error), indică resurse sau directoare care merită o investigate ulterioara (conform OWASP)) sunt colorate diferit pentru a fi evidențiate în rândul celorlalte cereri. Aceste cereri cu posibil succes au posibilitatea de a fi folosite ulterior în aplicație, folosind Click dreapta, observăm următoarele.opțiuni:

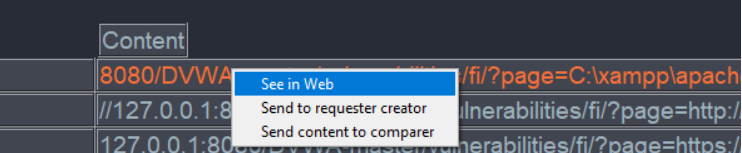


Fig 4.2 Opțiuni valabile input-urilor testate

Prima opţiune ne permite vizualizarea in cadrul paginii web într‑un mod rapid, însă trebuie să fim atenţi să avem aceleaşi cookie-uri și/sau date de login, dacă sunt prezente, altfel riscăm să interpretăm greşit rezultatele oferite de fuzzer. A doua opţiune ne redirecţionează request‑ul într-o pagină personalizată, unde putem să continuăm investigațiile, eficient și rapid. In paginile ce urmează vom prezenta acea funcționalitate mai în detaliu.Ultima opțiune ne copiază output-ul response-ului într­-o pagină specială care com­pară două texte. Această funcționlitate poate fi utilă când dorim să vedem dacă server-ul a reacționat diferit la două atac-uri, sau nu. În caz negativ, am putea obține două tipuri de vulnerabilități diferite.

* Al treilea submeniu este la randul ei formată din 4 componente, care ajută la definirea request-urilor, al setărilor HTTP și al definirea de fișiere payload. Prin adăugarea Cookie-urilor, ne asigurăm că request-urile nu sunt respinse de server înainte de investigarea dorită. Cookie­-urile se pot obține din javas­criptul paginii web apăsând F12, și scriind *document.cookie* în cmd-ul pag­inii. Din ce în ce mai multe site-uri conțin cookie-uri și setarea acestora devine o necesitate pentru acele site-uri. Cookie-ul poate fi setat atât din aplicație, cât și prin trimiterea unui fișier care conțien acel cookie.

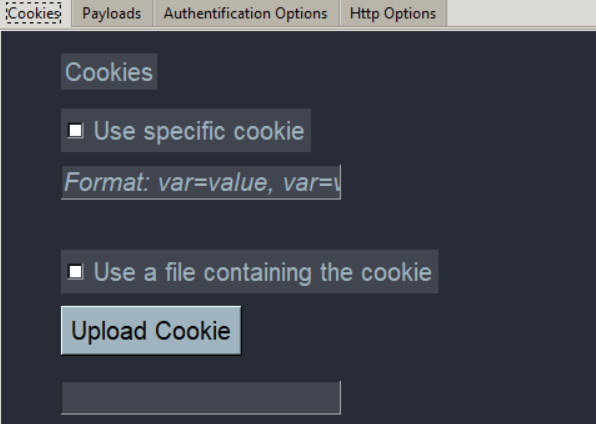
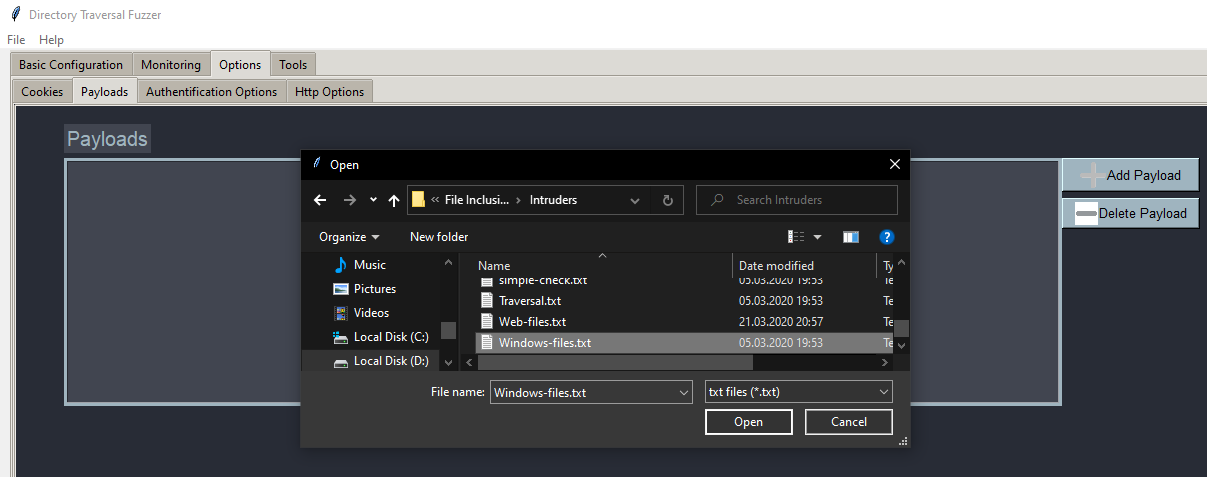


Fig. 4.3 Fereastră pentru setarea cookie-urilor

Request-urile primesc ca parametru un CookieJar. Mai jos vom vedea cum parsăm acest string de la utilizator și îl convertim în acest obiect, CookieJar, simplificat.

def set\_cookies(self, string):  
 keys\_and\_values = string.split()  
 self.cookie = requests.cookies.RequestsCookieJar()  
 for k in keys\_and\_values:  
 self.cookie.set(k.split('=')[0], k.split('=')[1])  
 return self.cookie

De asemenea putem adăuga câte fișiere de payload dorim, fiecare linie din fișier fiind tratată ca un string.



În fereastra de *Authentification* Options fuzzer-ul trimite un request de tipul *GET* prin care se autentifică cu datele aferenteintroduse de utilizator*.* Aceste date de autentificare se adaugă la fiecare request din sesiune, şi ne ajută să accesăm pagini prin request-uri, unde logarea este necesară.

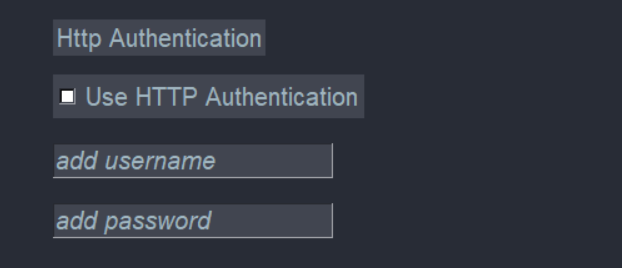


Fig. 4.4 Fereastră pentru setarea autentificării

Această funcționalitate se obține foarte repede, folosind modulul requests:

def set\_auth(username, password, login\_page):  
 global req\_Session  
 from requests.auth import HTTPBasicAuth  
 req\_Session.get(login\_page, auth=HTTPBasicAuth(username, password))

Nu este necesar să transmitem ca parametru pagina de login al serverului, deoarece, dacă incercăm să trimitem într-un request o pagină care necesita logarea, response-ul nu ne întoarce nicio eroare, ci ne întoarce pagina de login aferentă. Deci este suficient să trimitem un request pentru a afla pagina de login al server-ului, facându-i utilizatorului viata mai ușoară.

A patra pagină “*Http Options”* ne oferă opțiunile de a seta timpul maxim de așteptare pentru fiecare request în parte. Această opțiune poate fi utilă în ca­zul în care serverul are un timp de răspuns prea mare și afectează substanțial viteza fuzzer-ului. Setarea timpului de așteptare este doar un parametru din obiectul de tip requests.

req = req\_Session.get(url, cookies=self.cookie, timeout=self.time\_per\_request)

Vorbind de viteză de lucru, biblioteca concurent\_futures ne asigură lucrul cu thread-uri într-un mod prietenos:

def get\_request\_status(self, fuzzed\_urls, method, root):  
 with concurrent.futures.ThreadPoolExecutor(max\_workers=self.threads\_no) as executor:  
 url\_list = {executor.submit(self.send\_request, method, url): url for url in  
 fuzzed\_urls[self.fuzz\_counter:]}  
 for future in concurrent.futures.as\_completed(url\_list):  
 fuzzed = url\_list[future]  
 try:  
 status = future.result()  
 if status not in self.request\_statuses:

unde *fuzzed\_urls* este lista de url-uri cu potențial malițios, *threads\_no* fiind numărul de thread-uri dat de utilizator, iar *self.send\_request* metoda care trimite cererile http, aceasta returând codul statusului http rezultat, pe care îl analizăm ulterior.

De asemenea putem adăuga headere http personalizate pentru fiecare request din sesiune, și putem folosi de asemnea, un user agent fake.

În Python, conceptul de pereche, este definit ca și *tuple*. Header-urile unui request sunt niste perechi de tipul nume-valoare. Mai jos vedem cum pentru variabila req\_Session, acel obiect de sesiune al request-urilor, adăugăm diferite header-uri:

def set\_http\_headers(self):  
 global req\_Session  
 headers\_tuple = {}  
 for tuple\_ in self.http\_headers:  
 headers\_tuple[tuple\_[0]] = tuple\_[1]  
 req\_Session.headers = headers\_tuple

Mai jos putem observa imaginea cu interfața grafică care se ocupă cu aceste header-uri HTTP.

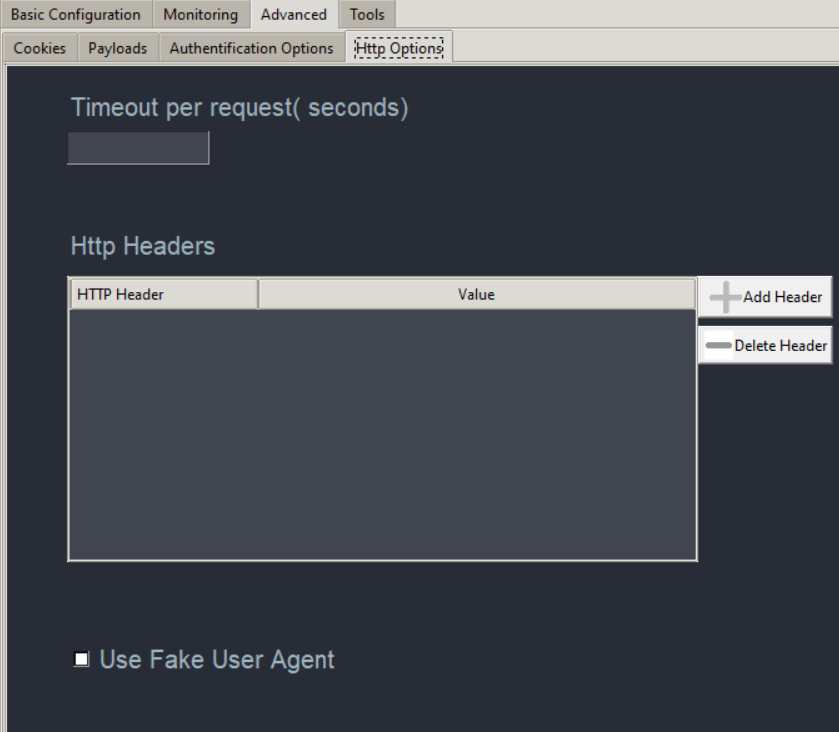
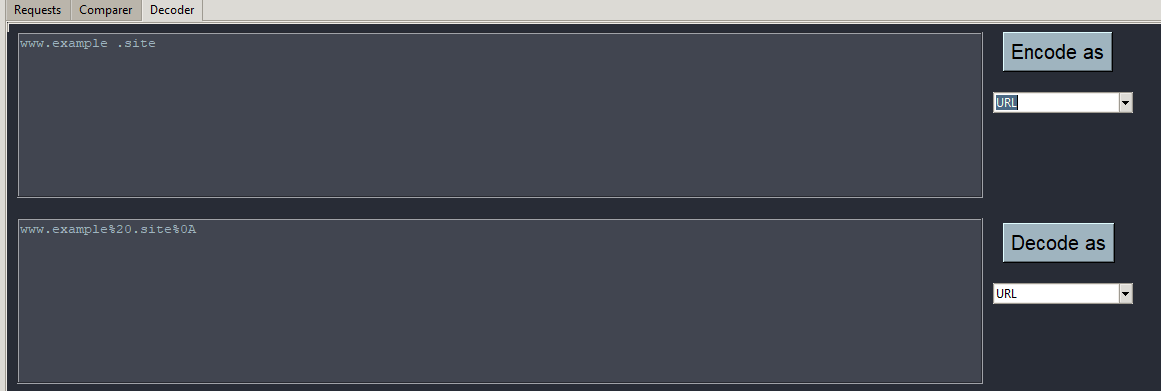


Fig. 4.5 Fereastră pentru adăugare alte setări http

* Al patrulea submeniu, *Tools* conține trei funcționalități. *Decoder*, unde putem coda și decoda diferite string-uri în diferite tipuri de date(html, base64, etc.) pe care ulterior le putem adăuga in request-urile, să extragem anumite date in response, etc.

Fig. 4.6 Funcționalitatea decoder al proiectului



În exemplul de mai sus am codat un string de forma unui link folosind URL encoding. Observăm că spațiul este înlocuit cu *“%20”,* iar terminatorul de sir cu *“%0A”* .Decodând vom ajunge la string-ul inițial.

A doua funcționalitate al acestui submeniu este comparer, care semnalizează prima linie în care găsim o diferență între texte. Aceasta este utilă când dorim sa comparăm răspunul serverului la două request-uri diferite. Dacă textele sunt diferite, este posibil ca să găsim o vulnerabilitate, datorită unei securității “hardcodate”.

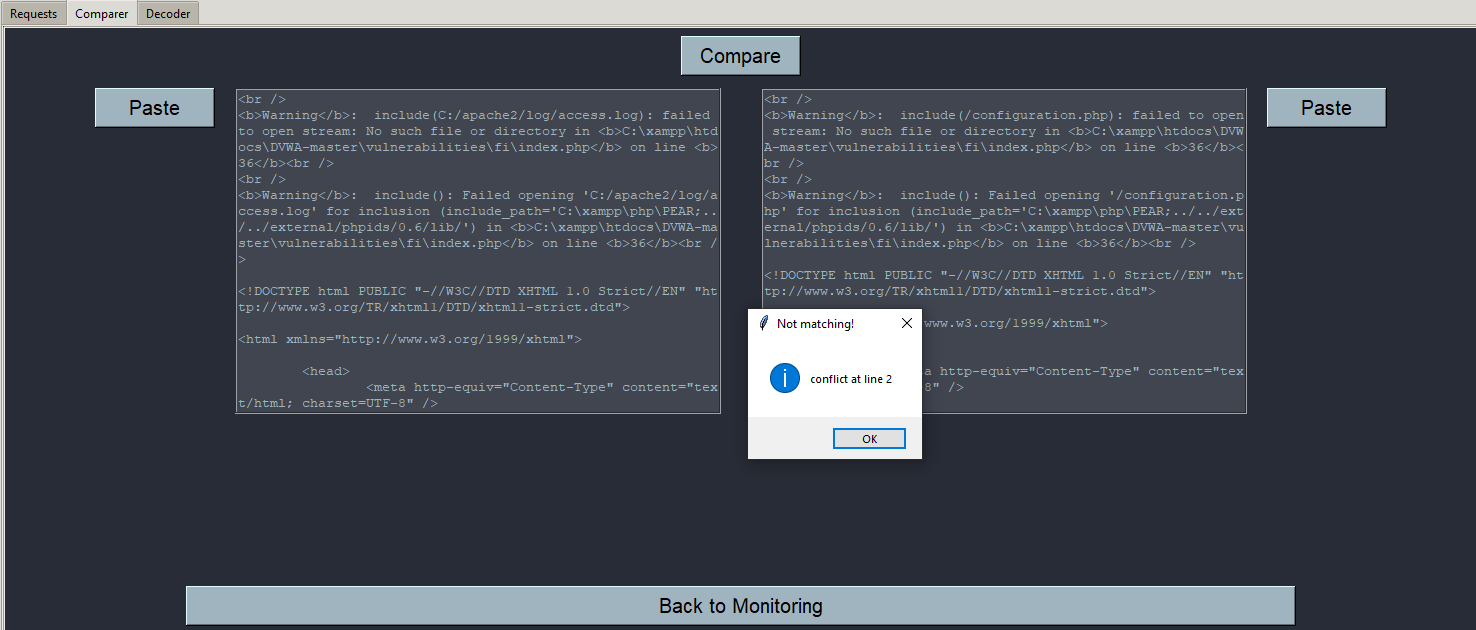


Fig. 4.7 Funcționalitatea comparer al proiectului

A treia funcționalitate este data de fereastra *Requests* ,cu scopul de a ne crea propriile header-uri în request, pe baza site-ului pe baza setărilor pe care le-am menționat anterior în aplicație. În dreapta putem observa răspunsul serverului, și de asemenea avem opțiunea de a copia textul, pentru a-l obține mai ușor în pagina *Comparer*, și de asemenea putem vedea răspunsul în pagina de Web al Chrome-ului.

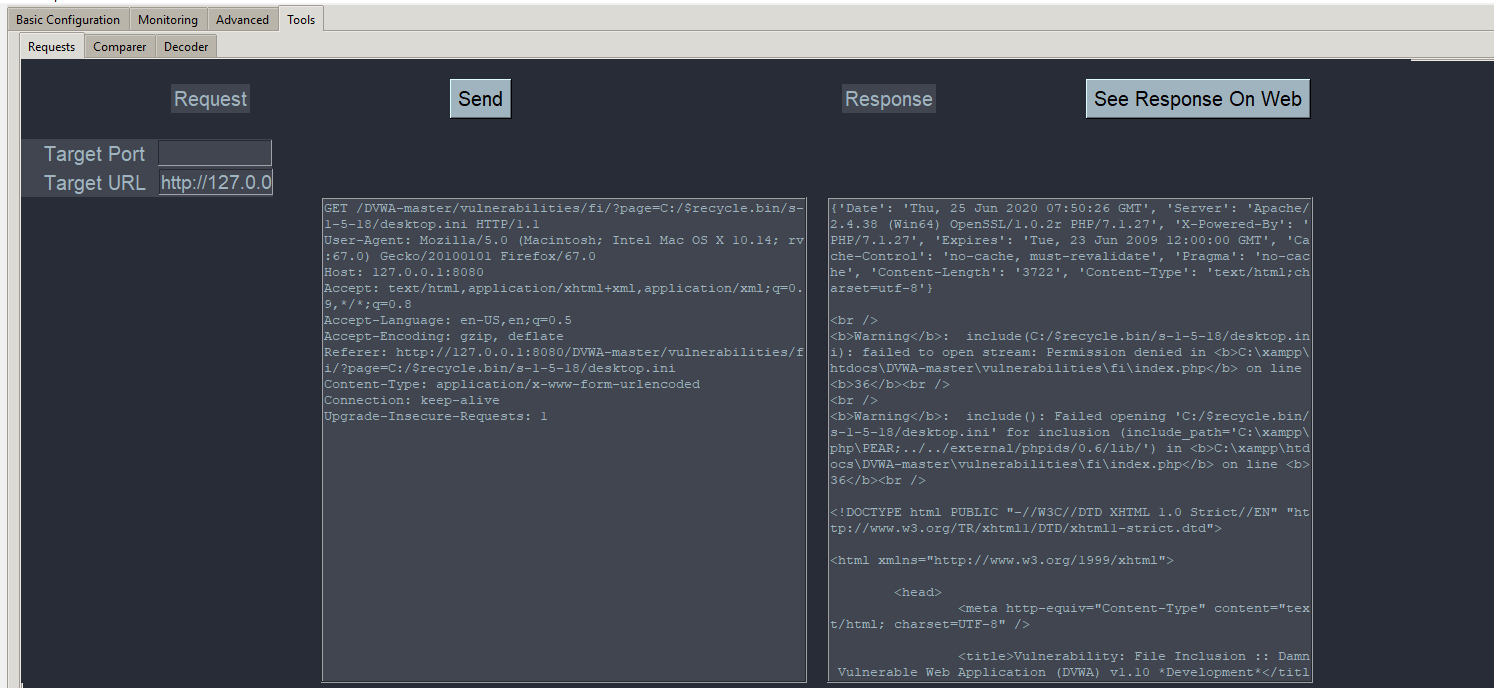


Fig. 4.8 Funcționalitatea requests al proiectului

## Tehnologii folosite pentru implementare

**Tehnologia Python**

Python este un limbaj de programare de nivel înalt, apărut in 1991, care utilizează folosit în majoritatea aplicațiilor importante din contextul actual, de la sisteme de operare si platforme hardware, la generarea dinamica de continut (in paginile Web), motoare de cautare , cel mai puternic fiind Google ,bioinformatică, etc.

Limbajul este folosit de asemena pentru aplicații software de scară largă, precum serverul de aplicatii Zope si sistemele de file-sharing Mnet si BitTorrent. Statisticile arată ca “traficul efectuat pe Internet este in proportie de 35% realizat de traficul BitTorrent”[16]. Motorul de căutare Google este top 1 in lume referitor la traficul internațional zilnic, motor de căutare programat în python.

Python este un limbaj multi-paradigma, care poate lucra pe mai multe stiluri de programare: programare functionala, orientata obiect, structurata si orientata-aspect. Python este un limbaj de nivel foarte inalt, care are incorporat un shell pentru depanarea programelor de dimensiuni mari precum si de un mecanism de tratare a erorilor superior majorităților limbaje de programare.

Tipurile de date ale limbajului Python au un caracter general, ceea ce oferă un avantaj mare și permite prelucrarea și efectuarea de calcule asupra acestora într-un mod mai apropriat de perceptia umană, spre deosebire de alte limbaje. Fiind un limbaj de nivel foarte inalt, eficienţa şi minimizea lungimii codului sunt superioare, pentru că tipurile de date ale Python permit realizarea de operatii complexe intr-o singură/mult mai puține instructiuni decât un alt limbaj care se bazeaza pe orientarea pe obiecte.De asemenea, lipsesc acoladele, fiind înlocuite cu tab-uri specifice fiecarei caracteristici ale operatiei respective. Lipsește și declararea de variabile, care salvează de asemenea mult spațiu în cod. In plus, limbajul este complet extensibil, interpretorul Python putand de exemplu fi adaugat intr-o aplicatie C/C++ si folosit ca limbaj de extensie sau de comandă pentru acea aplicație.

Limbajul a fost structurat astfel încât să fie foarte usor lizibil. Spațierea sugerează o regulă necovențioanală de prioritizare a instrucțiunilor. Foloseste ca instrucțiuni de cod cuvinte cheie din limba engleza în timp ce alte limbaje folosesc punctuatia, si conține mai putine constructii sintactice decat majoritatea limbajelor (C/C++, Pascal sau Perl).

Interpretorul Python dispune de un shell, unde se poate executa cod python într-o manieră mult mai rapidă și practică. Pe lânga shell, se poate lăuda cu un framweork ușor de utilizat ce se ocupă cu debuggul aplicațiilor.

In concluzie, Python este un limbaj universal care poate fi folosit atat în progamarea strcuturata, poate crea programe pentru diverse sisteme de oeprare, este utilizat în inteligență artificiala, în bioinformatică/ biometrie, la programarea orientată pe obiect, ale marilor aplicații,la generarea dinamicăa de conținut, etc. Limbajul Python asigură o flexibilitate crescută în ceea ce privesțe dezvoltarea unei aplicații, în propriul stil de programare folosind resurse proprii.

## 4.5 rezultatele testelor efectuate și interpretarea acestora

Pentru o mai bună prezentare si interpetare a rezultatelor, am ales ca aplicație de test ***DVWA*** (Damn Vulnerable Web Application), o aplicație vulnerabilă la diverse tipuri de atacuri, incluzând subiectul de interes, si anume, atacuri de tip “Directory Traversal”. Această aplicație are mai multe trepte de securitate(low-medium-high-impossible). Vom face o scurt rezumat pentru fiecare treaptă, pentru a observa cum răspunde fuzzer-ul la diferite niveluri de securitate. Testele vor fi executate in cadrul aceluiași server ( găzduit in hostul local de către **XAMPP,** careeste un “pachet de programe free software, open source și cross-platform web,server,care,constă,în Apache,HTTP, Server,MySQL database și interpretoare pentru scripturile scrise în limbajele de programare PHP și Perl”.[17]

Pentru nivelul low nu găsim niciun protocol de securitate, niciun filtering nu este utilizat, deci putem folosi atât LFI (Local file inclusion) cât și RFI(Remote file inclusion). Folosind setările de bază, și adăugănd un fișier de payload pentru Windows, deoarece acesta este sistemul de operare pe care rulează exploit-ul nostru, obținem următoarele rezultate.

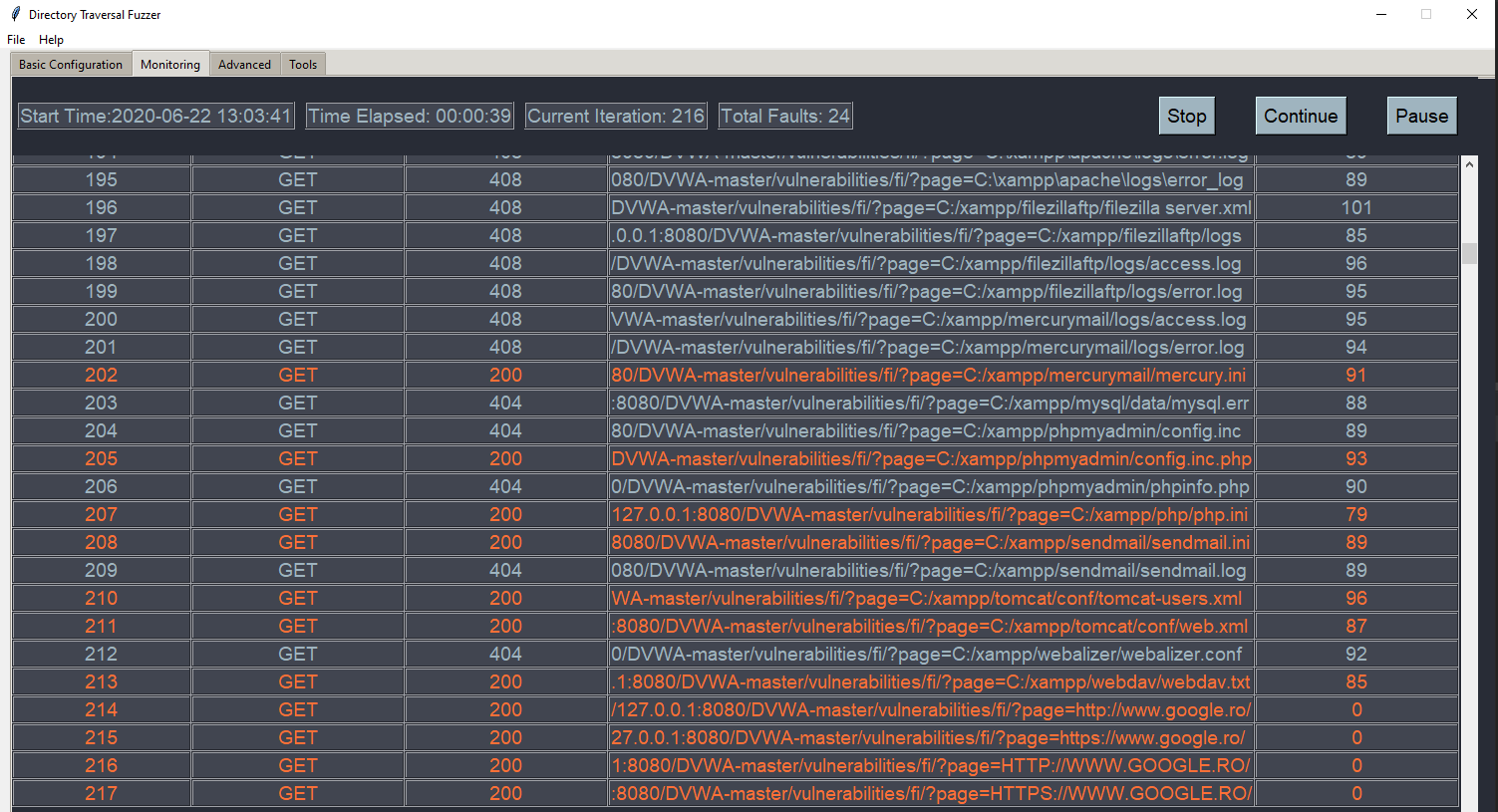


Fig. 4.9 Rezultatele obținute de fuzzer

In partea de sus se observă timpul durat procesului de fuzzing, numărul de atacuri și, indreapta, cele cu succes, de asemenea putem opri procesul pentru a obți statisticile pentru rezultatele aferente perioadei respective de timp.

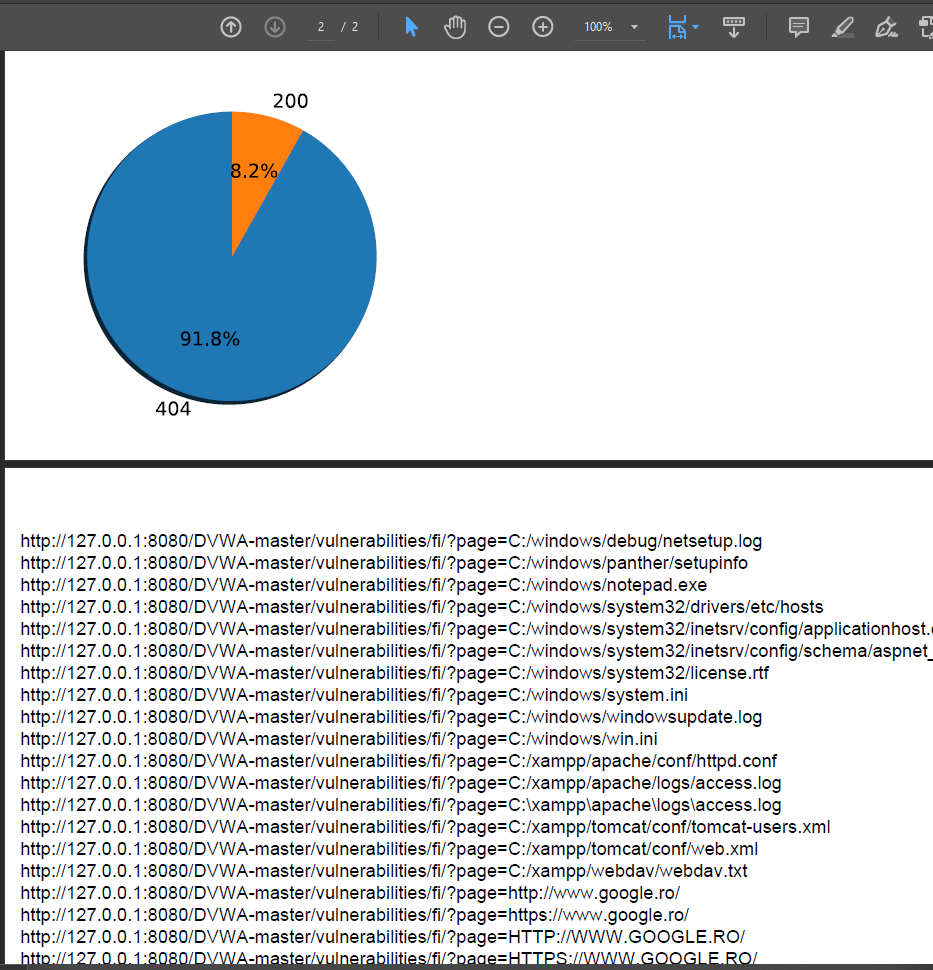
Observăm că rezultatele pozitive cu status 200 sunt colorate diferit, pentru a fi ușor vizibile. În cadrul acestora avem diferite opțiuni, cel mai frecvent fiind de a vedea rezultatul in web, de a trimite request-ul într-un window al aplicației care trimite request-uri personalizate. Acesta paote fi foarte util în cazul in care dorim să continuăm cercetările pentru un anumit rezultat, mărind eficiența și acuratețea atacurilor ( Fig 4.8).

Mai avem opțiunea de a compara rezultatele unor request-uri, pentru a verifica dacă serverul a răspuns diferit sau nu la diferite request-uri trimise către acesta. Dacă au răspuns diferit e o posibilitate de a găsi o vulnerabilitate nouă pentru acel exploit.

De asemenea, putem selecta optiunea de generarea a unui pdf unde găsim nişte statistici referitoare la sesiunea de fuzz respectivă si anume:

* Un grafic care reprezintă numărul de atacuri cu succes pe timpul perioadei de fuzzing
* Un grafic sferic ce reprezintă rapoartele de atacuri de tip *Payload*, adică date preluate din fisier, *Mutators*( url-uri alterate prin aplicarea unor metode specifice lucrurilor pe biti, adăugarea de string-uri malitioase, etc) si Depth\_Search(adăugarea de caractere speciale pentru a adăuga niveluri de căutare in radacina directorului serverului atacat).Graficul apare în cazul în care a fost selectată cel puțtin una dintre cele doua opțiuni,
* Proporția codurilor status rezultate în urma cererilor HTTP trimise către server.

. De exemplu, pentru modelul anterior pdf-ul generat arată in felul următor:



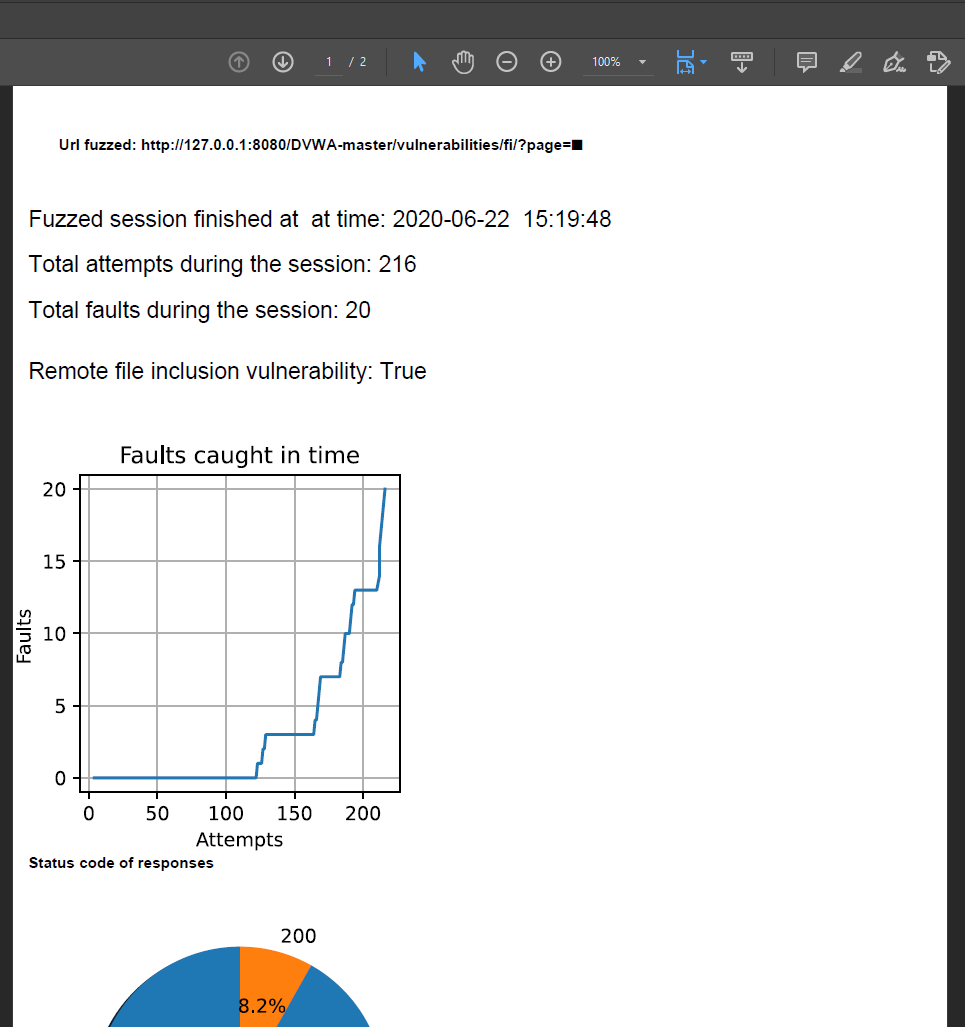


Fig. 4.10.0, Fig. 4.10.1 Informații generate de fuzzer potrivit testelor anterioroare

Chiar dacă niciun fel de securitate este prezentă, rezultatele pot să difere în funcție de structura de fișiere locală a fiecărui server, în funcție de payload-ul ales, de nivelul de adâncime în căutare, in cazul căilor de fisier relative, în funcție de sistemul de versiunea sistemului de operare, etc.

În cazul securității medie, singura diferență este că request-urile ce include “http” sunt sterșe din pagină, drept urmare tehnica RFI este afectată, insă putem folosi “HTTP” ca să ‘păcălim’ securitatea aplicației. De asemenea ‘../’ si ‘\..’ sunt şterse din pagina cerută, cu scopul de a elimina incercarea unei căutare din rădăcină a căilor relative, insă putem folosi căi absolute.



Fig 4.11 Implementarea aplicației vulnerabile DVWA

Acesta este un bad practice care poate duce la pierderea confidentialității unor date importante, folosind fuzzerul si setările potrivite.

In exemplul anterior, pentru simplitatea lucrurilor, nu am folosit căi relative, ci doar căi absolute pentru a imbunătăți vizualizarea request-urilor generate. Căutând in adâncimea de 5 directoare al fisierului rădăcină, folosind și caractere speciale, care pot trece de securitatea unui server poate consta în următorul exemplu:

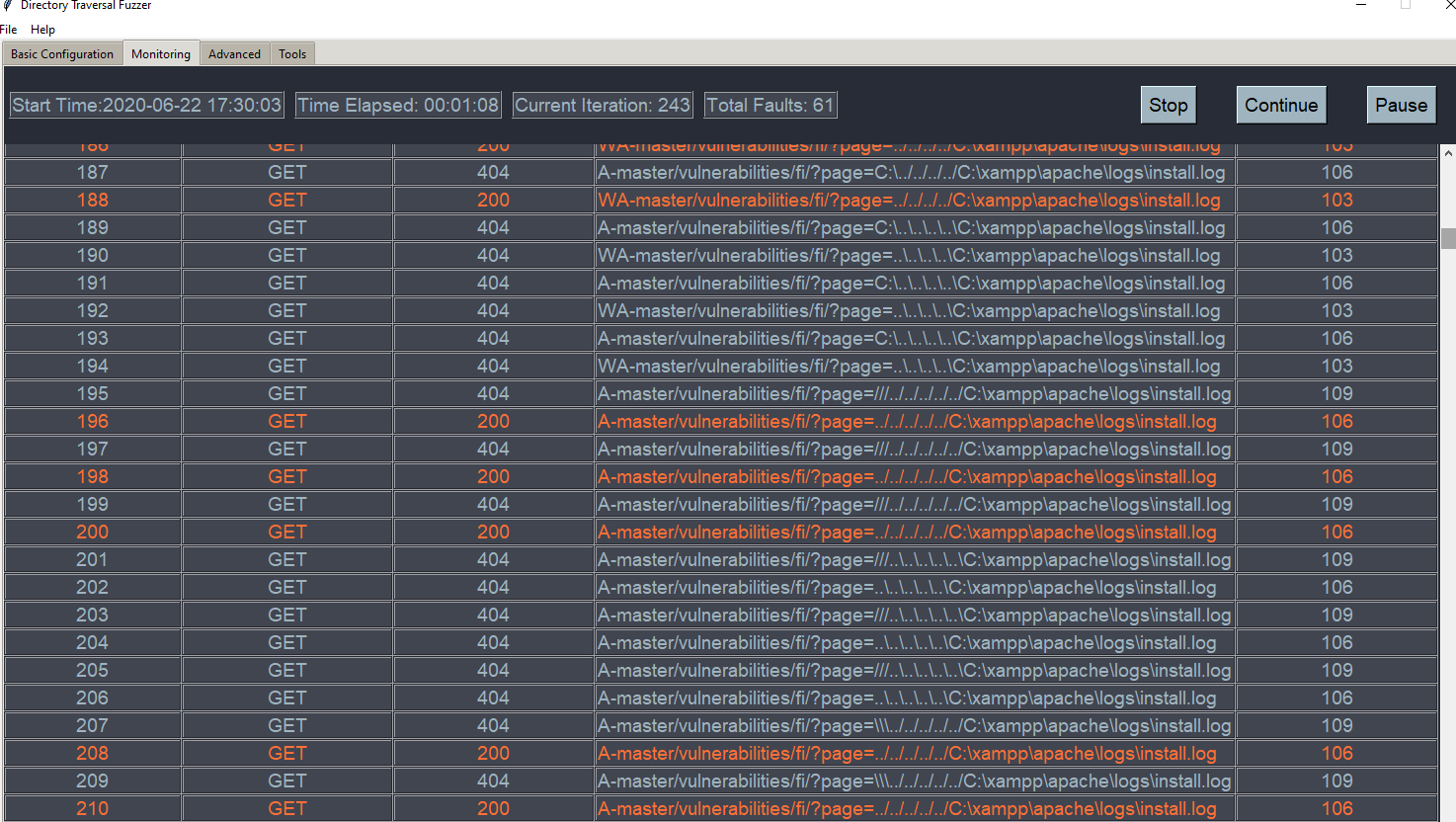


Fig. 4.12 Rezultate obținute de fuzzer folosind căutarea in adâncime

Se poate observa că, deși am căutat in adâncime, am reușit să obținem rezultate pozitive, dat fiind faptul că am folosit căi absolute, iar cum căutarea în adâncime este blocată (ștearsă din url) de nivelul mediu al DVWA, este ca și cum am fi testat folosind valorile anterioare (vezi Fig 4.9).Deși, în alte situații, poate fi folositoare, depinde de arhitectura și securitatea serverului web țintă.

## 4.5.1 Compararea rezultatelor dintre fuzzerul anterior si Burp Suite Intruder

Nu putem utiliza localhost-ul ca host pentru aplicația DVWA deoarece Burp Suite Intruder folosește acest port. Așa că vom testa aplicația DVWA hostată online, de către [www.hack.me](http://www.hack.me) folosind adresa corespunzătoare vulnerabilității directory traversal, din cadrul acesteia, în modul de securitate *medium*, de care am discutat la inceputul capitolului curent. După cum spuneam, pentru a intercepta pagina pe care vrem să facem testele, activăm opțiunea Intercept al Burp-ului, și dăm refresh paginii dorite.

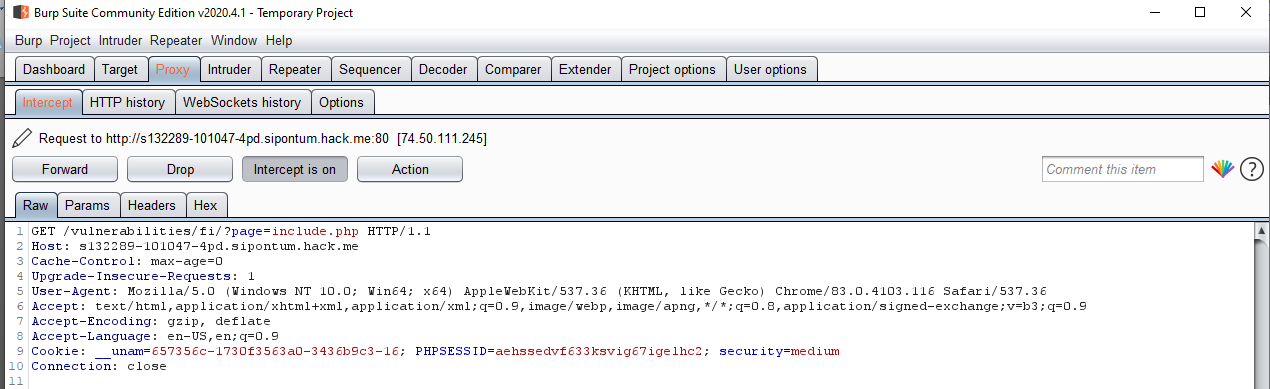


Fig. 4.13.0 Interceptarea datelor folosind Burp Suite

Următorul pas este setarea fișierului de payload, pe care îl configurăm din submeniul Payloads al Intruder-ului.

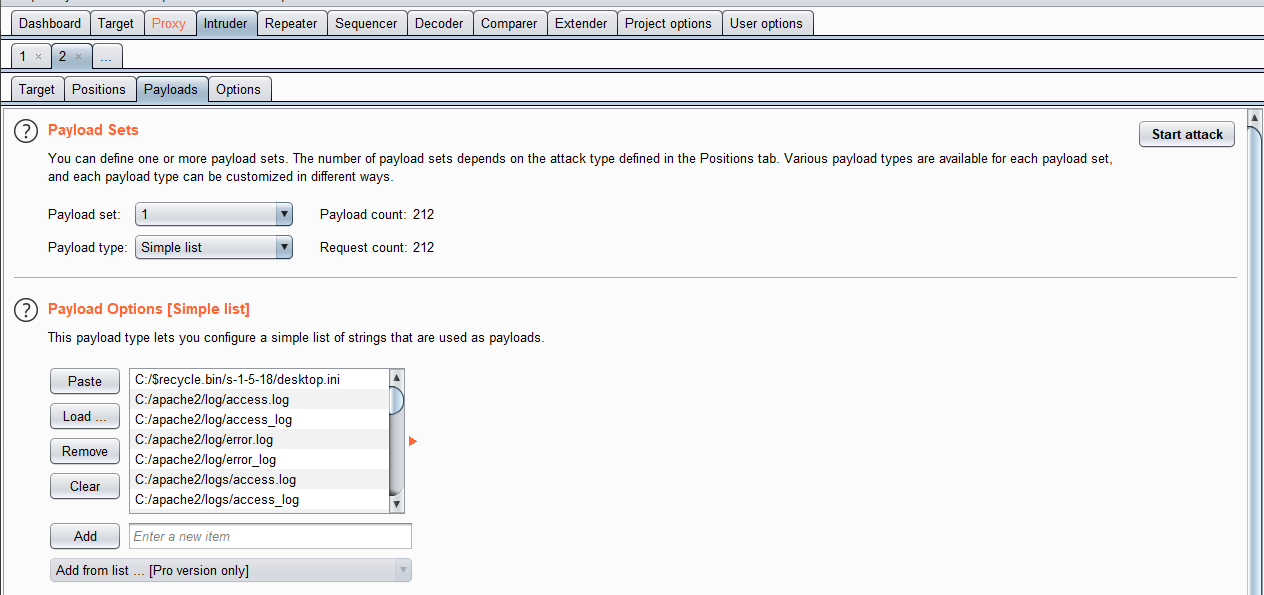


Fig. 4.13.1 Setarea payload-urilor folosind Burp Suite

După care, apăsam start și urmărim rezultatele testului. Făcând aceleași setări pentru fuzzer-ul propriu, haideți să vedem ce diferențe apar între fuzzere.

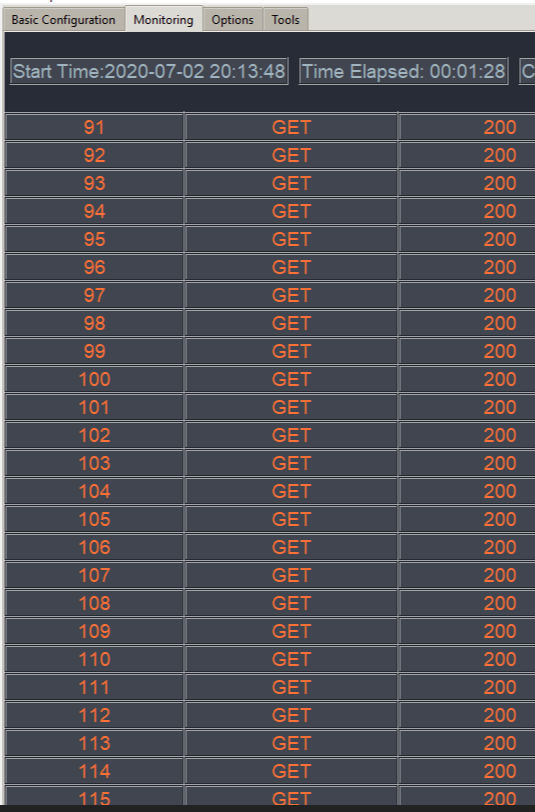
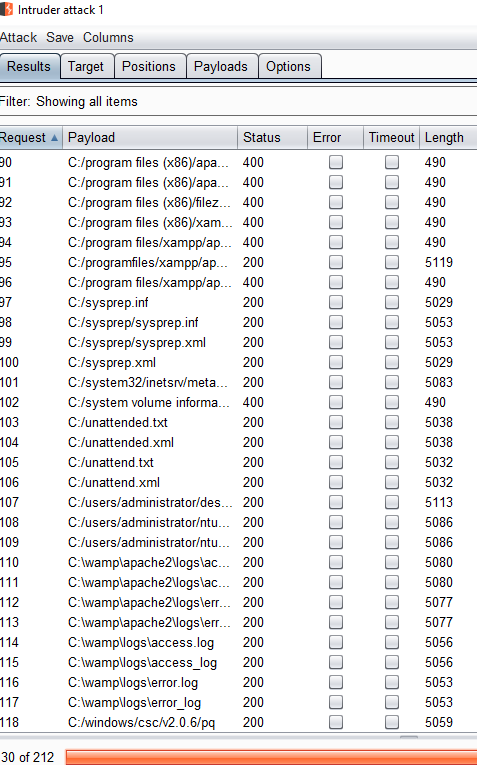


Fig 14.0, Fig 14.1, rezultatele obținute de Burp Suite, respectiv fuzzerul propriu

Se observă că pentru Burp Suite(poza din stânga) apar câteva status-uri de 400, care, în fuzzer-ul nostru sunt statusuri OK(200), în locul unde input-ul trimis în request nu este în format URL ci conține spații, si eventuale caractere speciale. Motivul este că librăria requests din Python convertește automat string-ul dat ca input în cod URL.Spre exemplu, spațiile le înlocuiește cu “%20”, etc.Desigur, și Burp Suite conține acestă opțiune. Dacă am fi bifat această opțiune, rezultatele ar fi fost exact la fel.

În termeni de viteză de lucru, varianta free al Burp Suite-ului este mult mai inceată decât a fuzzer-ului propriu. Varianta free nu permite setarea numărului de thread-uri, a timeout-ului pentru fiecare request, etc.) Pentru 212 de payload-uri, Burp Suite a utilizat aproximativ 5 minute, în raport cu 1:28 minute ale aplicației proprii. Însă, per total Burp Suite oferă opțiunea de a crea și edita reguli pentru adăaugarea payload-urilor în fișiere, și multe alte tool-uri pentru diferite tipuri de testare, fiind cu siguranță unul dintre cele mai bune programe pentru fuzzing-ul în cadrul aplicațiilor web.

# Concluzii

## Sinteza principalelor idei din lucrare

La începutul lucrării am văzut ce reprezintă vulnerabilitățile, ca fiind niște erori de programare și dezvoltare, care pot apărea în diferite etape de dezvoltare ale aplicației. Am mai analizat una dintre clasificările relevante ale vulnerabilităților (apărare defectuoasă, gestiunea slabă a resurselor și conexiune instabilă), și am exemplificat metodele celor două tehnici de detecție ale acestora. Tehnicile statice accesează în mod direct codul sursă fără a rula aplicația, în comparație cu tehnicile dinamice, care rulează codul sursă al acesteia.

În continuare ne-am concentrat pe vulnerabilitățile aplicațiilor web, unde cu ajutorul OWASP, organizația non profit care oferă unele dintre cele mai “preţioase” date legate de securiatea aplicaţiilor, am prezentat principalele vulnerabilități (SQL Injection fiind cea mai cunoscută și utilizată metodă, lucru destul de evident deoarece toate site-urile oferă informații, care sunt stocate în baza de date).

Următoarea idee principală a fost să introducem conceptul de fuzzing, concept black box prin care se trimit input-uri cu potențial malițios aplicației într-un mod automat. Clasificarea și arhitectura fuzzer-elor web au reprezentat următorul reper, urmat de o comparație între avantajele și dezavantajele utilizării fuzzer-elor în contextul securității aplicațiilor web, pentru a observa importanța acestui tip de testare în securitatea Web. Cu toate informațiile prezentate anterior, lucrarea practică nu a mai reprezentat o noutate, și am putut face o descriere a modului de funcționare și utilizare al fuzzer-ului. Ulterior, am generat niște teste și am comparat din diferite puncte de vedere funcționalitățile acestuia cu versiunea free al Burp-Suite-ului.

În urma celor prezentate, și al implementării practice, sunt de părere că fuzzing-ul este o parte importantă a securității web, deoarece omului ii este foarte greu să identifice erorile si bugg-urile în timpul în care un fuzzer o poate face, automat, în cadrul a mai multor domenii, discutate pe parcursul lucrării. De asemenea, fuzz-testing-urile se pot efecuta în fiecare etapă a dezvoltării aplicației, și cu cât timpul alocat testelor este mai mare, cu atât securitatea aplicației crește. Fiecare fuzzer poate fi eficient sau mai puțin eficient în comparație cu alt fuzzer cu funcționalități asemenăntoare, de aceea este recomandată utilizarea mai multor fuzzeri pentru menținerea securității ridicate. Chiar dacă fuzzing-ul nu poate asigura decât o proporție a securității generale a aplicației, consider că această tehnică trebuie inclusă în fiecare analiză a testării aplicației, pe lângă celelalte metode de testare, fiind automat, relativ ușor de utilizat, acoperând o serie mare de vulnerabilități, pot salva dezvolatorilor multe resurse.

## Direcții pentru continuarea cercetării

Prioritatea vulnerabilităților în cadrul aplicațiilor web diferă de la o perioadă la alta. Atacurile devin din ce în ce mai complexe, securitatea crește, structura server-elor și al sistemelor de operare este într-o continuă dezvoltare, de aceea pentru rezultatele optimale trebuie să fim la curent cu toate aceste aspecte, și să aplicăm teste de verificare a securității mergând pe premisa că aplicația este permanent, amenințată de atacuri cibernetice care pot pune probleme mari.

Continuarea cercetării pot include următoarele idei:

* Realizarea unui scenariu de exemplificare al utilizării fuzzer-elor web într-o situație mai complexă
* Exemplificarea informațiilor pe care le pot oferi diferite fișiere importante din sistemele de operare

# 6. BIBLIOGRAFIE

[1] Tobz Clarke, “Software Vulnerabilties”, in *Fuzzing for software vulnerability discovery*, Royal Holloway, University of London, 17 Februarie 2009

[2] Willy Jimenez, Amel Mammar, Ana Cavalli *Software Vulnerabilities, Prevention and Detection Methods: A Review*  
[3] Edward J. Schwartz, Thanassis Avgerinos, David Brumley Carnegie Mellon , *All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)*,University Pittsburgh: PA

[4] OWASP Official Site,

Internet: <https://owasp.org/>

[5] Sucuri organization, *OWASP TOP 10 Security Risks & Vulnerabilties*

Availabe:<https://sucuri.net/guides/owasp-top-10-security-vulnerabilities-2020/>

[6] Acunetix, *Web Application Vulnerability Report*, 2020

Internet : https://www.acunetix.com/acunetix-web-application-vulnerability-report/

[7] “Logging and Monitoring”

Internet: https://www.hack2secure.com/blogs/insufficient-logging-and-monitoring--a-brief-walk-through

[8] , Ari Takanen, Jared D. Demott, Charles Miller, *Fuzzing for Software Securit Testing and Quailty Assurance,* Artech House, Norwood, MA, 2008

[9] “Robustness testing”, Wikipedia, 15 Iunie 2020

Internet: https://en.wikipedia.org/wiki/Robustness\_testing

# [10] Noam Rathaus, Gadi Evron, Robert Fly, Aviram Jenik, Yoav Naveh, Charlie Miler, *Open Source Fuzzing Tools,* Burlington, MA, 2007

[11] Fuzz Testing, Wikipedia, 19 Iunie 2020

Internet: http://en.wikipedia.org/wiki/Fuzz\_testing

[12] Zap Proxy, 2020

Internet: https://www.zaproxy.org/

[13]Daemon Computing, Wikipedia, 26 Iunie 2020

Internet: https://en.wikipedia.org/wiki/Daemon\_(computing)

[14] Infosec, „OWASP ZAP Proxy”, November 26, 2012

Internet: https://resources.infosecinstitute.com/owasp-zap

[15]  [Adam Greene](https://www.informit.com/authors/bio/0d3557bd-2fb4-4d90-be63-1cd97380cd51), [Pedram Amini](https://www.informit.com/authors/bio/ee0de464-e4cd-4de7-80f3-a9a990440ffd), [Michael Sutto](https://www.informit.com/authors/bio/9b148652-4ccf-4992-8d93-8c0e6f771d73)n. (Nov 16 2007) [*Fuzzing: Brute Force Vulnerability Discovery*](https://www.informit.com/store/fuzzing-brute-force-vulnerability-discovery-9780321446114?w_ptgrevartcl=Fuzzing+Frameworks_768663)*,* [On-Line]

Internet: https://www.informit.com/articles/article.aspx?p=768663&seqNum=4

[16] Ciprian Pungila, “Tehnologia Python”, 2006

Internet: <http://www.ciprianpungila.com/python/tehnologie.html>

[17] „XAMPP”, Wikipedia, 1 Iulie 2020

Internet: <https://en.wikipedia.org/wiki/XAMPP>