

# Faz testiranje

Seminarski rad u okviru kursa  
Verifikacija softvera  
Matematički fakultet

Luka Milošević, 1086/2019  
mi15167@alas.matf.bg.ac.rs

24. februar 2020

## Sažetak

Faz testiranje (eng. *fuzz testing*) je veoma efektivna tehnika za detektovanje sigurnosnih propusta u softverskim sistemima. Sastoji se iz prosleđivanja neočekivanih, nevalidnih ili delimično nasumičnih ulaza testiranoj aplikaciji što može dovesti do neočekivanog ponašanja. Ovakav pristup ne zahteva prethodno znanje o testiranom sistemu i dobro je prilagođen testiranju sistema zatvorenog koda. Negativne strane ovog pristupa su relativno visok nivo lažno pozitivnih rezultata (eng. *false positive*). U ovom radu prikazane su osnovne karakteristike faz testiranja i njene upotrebe.

## Sadržaj

<b>1</b>	<b>Uvod</b>	<b>2</b>
<b>2</b>	<b>Nastanak faz testiranja</b>	<b>2</b>
<b>3</b>	<b>Faz testiranje</b>	<b>2</b>
3.1	Faz testiranje metodom crne kutije . . . . .	4
3.2	Faz testiranje metodom bele kutije . . . . .	4
3.3	Zašto koristiti faz testiranje? Prednosti i mane faz testiranja	4
<b>4</b>	<b>Faze faz testiranja</b>	<b>5</b>
4.1	Generisanje test slučajeva . . . . .	5
4.2	Reprodukcija test slučajeva . . . . .	5
4.3	Propagacija test slučajeva kao ulaz programa . . . . .	5
4.4	Otkrivanje grešaka . . . . .	5
<b>5</b>	<b>Kvalitet faz testiranja</b>	<b>6</b>
5.1	Brzina . . . . .	6
5.2	Klasifikacija grešaka . . . . .	6
5.3	Redukcija test slučajeva . . . . .	6
5.4	Pokrivenost koda . . . . .	7
<b>6</b>	<b>Faz radni okviri</b>	<b>7</b>
<b>7</b>	<b>Zaključak</b>	<b>8</b>
	<b>Literatura</b>	<b>8</b>

## 1 Uvod

Testiranje je važan deo procesa razvoja softvera. Neke procene govore da se za testiranje i debugovanje utroši 50% ukupnog vremena [6]. Da bismo bili sigurni da softver radi ono što treba, i da to radi dobro, testiranje je neophodno. Kao zamena za testiranje nekad se nude metode formalnog dokazivanja. Iako one garantuju ispravnost, moguće ih je sprovesti samo na sisteme ograničene veličine i njihova potpuna automatizacija, za sada, nije moguća. Testiranje sistema se može automatizovati i ono je, sa strane isplativosti, jeftinije. S druge strane ono ne može da garantuje ispravnost, samo da poveća stepen pouzdanosti sistema. Softverski sistemi postaju sve važniji za sve vrste organizacija, ali i za individue. Samim tim raste i značaj njihovog kvaliteta. Ozbiljnost grešaka, koje se javljaju u softveru, može biti bilo gde na skali od malih vizualnih problema do pada sistema. Neke od njih mogu da dovedu do opasnih situacija za ljude. Ako za primer uzmemo sisteme koji se koriste pri lansiranju raketa u svemir, jasno je da i male greške u sistemu vode do ozbiljnih posledica. Iz svega ovoga proizilazi i važnost testiranja softvera. Na žalost, postoji mnogo problema u vezi sa testiranjem. Ono je komplikovano i oduzima dosta vremena. Uopšteno, komplikovane zadatke obavljaju ljudi koji su skloni greškama. Ovo se može popraviti automatizacijom [5].

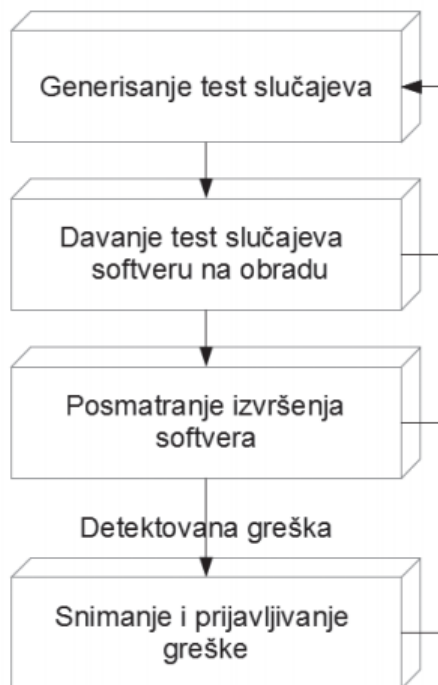
## 2 Nastanak faz testiranja

Kako bi softver bio siguran potrebno je testirati njegovu sigurnost. Jedna od metoda je nazvana testiranje faz metodom. Tehniku je uspostavio Prof. Barton Miller sa Univerziteta u Wisconsinu 1988. kao studentski zadatak nazvan: “Operating System Utility Program Reliability – The Fuzz Generator“. Osnovna ideja testiranja faz metodom je generisanje slučajnih ili pseudo-slučajnih podataka kao ulaz u sistem. Ukoliko sistem prestane da funkcioniše nakon nekog ulaza onda se taj niz podataka zapamti za dalju analizu. Danas se testiranje faz metodom najviše koristi u velikim sistemima gde se vrši testiranje na principu crne-kutije, zato što se pokazalo da faz metoda daje jako dobar odnos cene i vremena naspram kvaliteta testiranja. Mogućnosti primene su jako velike tako da se može testirati vrlo široki spektar različitih sistema od web-aplikacija, funkcija u kodu i sl [1].

## 3 Faz testiranje

Faz testiranje ili negativno testiranje predstavlja efikasnu metodu za otkrivanje sigurnosnih propusta, posebno u slučajevima aplikacija zatvorenog koda. Ovaj način testiranja predstavlja automatizovano ili delimično automatizovano testiranje brojnih graničnih slučajeva u softveru, pri čemu se kao ulazni podaci biraju nasumični, delimično nasumični ili delimično nevalidni podaci. Sistem koji vrši faz testiranje se naziva fazer (eng. *fuzzer*). Usled visokog stepena automatizacije, ovaj vid testiranja ne zahteva mnogo ekspertskog vremena. Faz testiranje se sastoji od generisanja test slučajeva ulaznih podataka i praćenja izvršenja testiranog softvera u toku obrade. Koraci pri faz testiranju su prikazani na slici 1.

Faz testiranjem se obično detektuju programske greške koje dovode do korumpiranja struktura u memoriji. Najrasprostranjeniji oblik ove vrste grešaka je prelivanje bafera. Postoji više različitih pristupa faz testiranju,



Slika 1: Koraci pri faz testiranju

a najznačajnija su: faz testiranje bazirano na mutaciji i faz testiranje zasnovano na generisanju. Mutaciono faz testiranja se bazira na mutaciji poznatih i validnih ulaza, a faz testiranje generisanjem se bazira na generisanju novih ulaza na osnovu modela validnih ulaza. Najprostiji oblik mutacije ulaza jeste takozvani *bitflipping* kod kojeg se nasumični bajtovi ulaza nasumično menjaju. Iako mutacioni faz sistemi zahtevaju manje znanja o konkretnoj aplikaciji koja se testira i lakši su za implementaciju, obično imaju slabu pokrivenost testiranog koda usled odbijanja mutiranih ulaza kao nevalidnih. Do ovoga može da dođe usled raznih polja sa kontrolnim sumama, pravilima za enkodovanje i logičkim proverama kojih faz sistem nije svestan jer ne poseduje model podataka nad kojim sistem operiše. Sa druge strane, generativni faz sistemi obično imaju bolju pokrivenost testiranog koda, ali se takođe mora napraviti verni i detaljan model ulaznih podataka. Ovaj model bi uključivao sva znanja o kontrolnim sumama i logičkim proverama koje aplikacija vrši kako bi sistem mogao da generiše ulazne test podatke koji ne bi bili odbačeni kao nevalidni u ranim fazama obrade. Pravljenje detaljnog modela ulaza može da predstavlja lak zadatak kada je specifikacija ulaza dostupna, ali i veoma težak zadatak u slučaju zaštićenih aplikacija. Za razliku od mutacionih faz testera koji se mogu primenjivati na širok spektar različitih aplikacija bez prevelike modifikacije, generativni faz testeri su primenljivi isključivo na relativno slične aplikacije. Jedan pristup rešenju ova dva problema: slabe pokrivenosti koda i potrebe za poznavanjem detalja podataka, jeste faz testiranje

u memoriji. Umesto generisanja novih ili mutacije postojećih podataka, faz testiranje u memoriji se fokusira na mutaciju podataka koji se već nalaze u memoriji procesa testirane aplikacije [7]. Faz testiranje može da koristi dve metode: metodu crne i bele kutije, od čega zavisi način generisanja ulaznih podataka [5].

### 3.1 Faz testiranje metodom crne kutije

Kada se testiranje vrši metodom crne kutije, ulazni podaci se generišu ili mutacijom nekog seta ulaznih parametara, ili korišćenjem nekog šablona (gramatike), koji opisuje set ulaznih podataka. Metod crne kutije je češće korišćen jer omogućava da se za kratko vreme kreira mnogo ulaznih podataka koji često uspevaju da otkriju veliki broj grešaka [5].

### 3.2 Faz testiranje metodom bele kutije

Pri korišćenju metode bele kutije, generisanje podataka koristi unutrašnju strukturu i dizajn sistema za generisanje parametara. Pri izvršavanju sistema sa prvim (zadatim) setom ulaznih parametara, prate se stanja u sistemu i generišu se ograničenja za parametre. Ova ograničenja se prave ispitivanjem svih uslovnih grananja u programu (npr. if/else grane). Najčešće, prikupljanje ograničenja se postiže simboličkim izvršavanjem sistema za testiranje [5].

### 3.3 Zašto koristiti faz testiranje? Prednosti i mane faz testiranja

Faz testiranje može da bude veoma korisno ali ne predstavlja srebrni metak<sup>1</sup> (eng. *silver bullet*). Faz testiranje pronalazi najozbiljnije bezbednosne propuste i greške, daje mnogo efektivnije rezultate kada se koristi u kombinaciji sa metodom crne kutije, beta testiranjem i ostalim metodama debugovanja. Često se koristi da pronađe nedostatke softvera. Veoma isplativa tehnika testiranja. Može se koristiti i za ilegalne radnje, gde hakeri mogu pomoću faz testiranja da pronađu ranjivost softvera [4]. Jedan od najpoznatijih bagova moguće je pronaći faz testiranjem *The Heartbleed Bug*<sup>2</sup>.

U nastavku će biti prikazane prednosti(+) i mane(-) faz testiranja [2]:

- + Obezbeđuje rezultate uz malo napora, jednom kada se faz testiranje pokrene, možete ga ostaviti satima, danima ili mesecima da biste našli greške bez interakcije.
- + Može otkriti greške propuštene pri ručnoj proveru.
- + Daje opštu sliku robusnosti ciljnog softvera.
- Neće pronaći sve bagove, testiranje može da propusti bagove koji će dovesti do potpunog pada programa, i može biti manje verovatno da pronađe bagove koji se pojavljuju u krajnje specifičnim okolnostima.

<sup>1</sup>Otkriće koje pruža najefikasnije rešenje određenog problema ili poteškoće, naročito ono koje je obično veoma složeno ili teško rešiti. Izraz se gotovo uvek koristi u smislu da takvo rešenje ne postoji.

<sup>2</sup>Reč je o ozbiljnom propustu u popularnom OpenSSL softveru koji omogućava krađu informacija zaštićenih SSL/TLS sigurnosnim protokolom. SSL/TLS je kriptografski protokol koji obezbeđuje sigurnost i privatnost komunikacije preko Interneta (web, email, instant poruke – IM i neke privatne virtuelne mreže – VPN) [8].

- Test slučajevi koji su pronađeni i koji dovode do pada programa mogu biti teški za analizu, zbog toga što faz testiranje ne daje previše informacija o tome kako se softver ponaša interno.
- Programi sa složenim ulazima mogu zahtevati mnogo više rada da bi se proizveo dovoljno dobar fazi test da bi se postigla što veća pokrivenost koda.

## 4 Faze faz testiranja

Da bi testiranje funkcionisalo efikasno, faz testeri svoje izvršavanje dele u različite faze. Fazeri često dele mnoge od tih zadataka u zasebne module. Na primer, imaju jednu biblioteku koja može mutirati podatke ili ih generisati na osnovu definicije i drugu za pružanje test slučajeva programu i tako dalje [2]. Ispod su detaljnije opisane faze faz testiranja.

### 4.1 Generisanje test slučajeva

Generisanje test slučajeva će varirati u zavisnosti od toga da li se upotrebljava faz testiranje zasnovano na mutaciji ili na generisanju. Granične vrednosti često umeju da budu izvor bagova, kao takve poželjno je dati im prednost i uključiti vrednosti kao što su veoma dugačke ili prazne niske, maksimalne ili minimalne vrednosti tipova u zavisnosti od računara, vrednosti kao što su -1, 0, 1 [2].

U zavisnosti od toga šta se testira, mogućnost postojanja određenih vrednosti ili karaktera koji verovatnije pokreću greške je velika neke od njih su: *null* vrednosti, karakter za novi red `\n`, vrednosti za formatiranje niski `%n`, `%s`, ....

### 4.2 Reprodukcija test slučajeva

Najjednostavniji način reprodukcije test slučajeva je pamćenje ulaza koji se koristi kada se otkrije pad programa. Međutim, postoje i drugi načini da se osigura ponovljivost koja u određenim okolnostima može biti prikladnija [2].

### 4.3 Propagacija test slučajeva kao ulaz programa

Propagacija test slučajeva kao ulaz programu često je jednostavno. Za mrežne protokole može uključivati slanje test slučaja preko mreže ili odgovor na zahtev klijenta. Za datoteka to može značiti izvršavanje programa sa argumentom komandne linije koji sadrži test slučaj [2].

### 4.4 Otkrivanje grešaka

Otkrivanje grešaka je kritično za faz testiranje. Ako ne može tačno da se odredi kada se program srušio, neće moći ni da se identifikuje test slučaj koji aktivira grešku. Postoji nekoliko uobičajenih načina da se tome pristupi:

**Korišćenje debagera (eng. *debugger*)** pruža najtačnije rezultate, gde je moguće dobiti putanju do greške. Međutim korišćenje debagera može značajno da uspori izvršavanje programa.

**Provera da li proces nestaje** umesto korišćenja debagera, jednostavno može se videti da li ID procesa i dalje postoji u sistemu nakon izvršenja test slučaja. Ako je proces nestao, verovatno je program pukao. Može se ponovo pokrenuti test slučaj pomoću debagera ako je potrebno još informacija o padu. To se može čak i automatski uraditi za svaki pad, i izbeći usporavanje debagera za svaki test slučaj.

**Vremensko ograničenje** trajanja programa gde se može postaviti vremenski period nakon kojeg se pretpostavlja da se program srušio ili zamrzao. Ovo takođe može otkriti greške zbog kojih program postaje nepouzdan, gde ne mora nužno da pukne. Program može normalno reagovati na test primere.

Bez obzira na metodu koja se koristi, program treba ponovo pokrenuti kad god se zaustavi ili se zamrzne, kako bi se omogućilo da se faz testiranja nastavi [2].

## 5 Kvalitet faz testiranja

Postoji nekoliko stvari koje mogu da se urade za merenje ili poboljšanje kvaliteta faz testiranja. Iako su ovo korisne stvari koje su bitne, moguće su situacije gde neće biti toliko bitne ukoliko se pronašlo dosta testova zbog kojih program pada [2].

### 5.1 Brzina

Možda je jedan od najvažnijih faktora pri faz testiranju je brzina. Koliko test slučajeva u sekundi/minuti se može pokrenuti? Što više test slučajeva se izvrši, veća je verovatnoća da će se u određenom vremenskom periodu pronaći grešku. Faz testiranje je nasumično, tako da je svaki test slučaj kao lutrija, a želimo što više test slučajeva koje je moguće dobiti u odgovarajućem vremenskom periodu pa brzina predstavlja jako bitan faktor [2].

### 5.2 Klasifikacija grešaka

Pronalaženje greške je samo početak procesa. Kada se pronađe test slučaj koji prouzrokuje pad programa, potrebno je analizirati grešku, otkriti šta je tačno greška, i na kraju popraviti je. U situaciji gde postoji preko hiljadu takvih test slučajeva ovo može biti jako iscrpljujuće.

Klasifikacijom grešaka može se odrediti koje greške su najzanimljivije i dati im prioritet pri rešavanju. Takođe moguće je i otkriti koji testovi prouzrokuju sličan bag i onda možemo zadržati samo testove koji prouzrokuju različite bagove.

Da bi to bilo moguće potreban je neki automatizovan proces koji nam pruža informacije o bagovima da bismo mogli da donesemo dobre odluke pri klasifikaciji.

Jedan od alata koji može pomoći je *!exploitable* firme Microsoft [2].

### 5.3 Redukcija test slučajeva

Kako faz testiranje menja ulaze nasumično, uobičajeno je da pored test slučaj koji prouzrokuje grešku postoji nekoliko varijanti tog test slučaja koji se razlikuju na delovima koji ne prouzrokuju grešku. Redukcija test slučaja je način smanjivanja test slučaja tako da ima što manje delova

koji ne prouzrokuju grešku. Ovakva redukcija može da se vrši ručno ili automatski pomoću fazera. Kada se pronade test slučaj koji prouzrokuje grešku fazer izvršava taj test nekoliko puta. Svaki put postepeno smanjuje promene koje su napravljene ali tako da se i dalje greška javlja pokretanjem tog test slučaja. Ovaj način može dosta da uprosti analizu greške tako što se fokus usmerava ka grešci [2].

## 5.4 Pokrivenost koda

Predstavlja merilo koliko koda je izvršeno od strane fazera. Ideja je da što je veća pokrivenost, program je bolje testiran. Merenje pokrivenost koda može biti teško i često zahteva binarnu instrumentalizaciju da bi se pratilo koji delovi koda su izvršeni. Pokrivenost koda može se meriti na različite načine kao što su: po liniji koda, po granama, po blokovima i putanjom kroz program. Pokrivenost koda nije savršena mera kada je u pitanju faz testiranje, jer je moguće izvršiti kod bez otkrivanja grešaka. Tu su takođe delovi koda koji se retko izvršavaju ili se ne izvršavaju uopšte kao što su proveru grešaka. Neki od alata koji se koriste za pokrivenost koda sadrže: *Pai Mei*, *Valgrind*, *DynamoRIO*, *DTrace* [2].

## 6 Faz radni okviri

Postoji veliki broj postojećih faz radnih okvira (eng. *fuzzing frameworks*) koji omogućavaju kreiranje fazera gde ne treba praviti ih od nule. Neki od ovih okvira su složeniji i potrebno je dosta vremena da se prilagode problemu koji je nama potreban. Takođe postoje i oni jednostavniji. Neki od okvira su: *Radamsa*, *Sulley*, *Peach*, *SPIKE*, *Grinder*, *NodeFuzz* [2].

Postoje i različiti fazeri koji već imaju namenu u faz testiranju. Neki od fazera su: *Google Sanitizers*, *afl-fuzz*, *Dizzy*, *dotdotpwn*, *Exploitable*, *Go-fuzz*, *iFuzzer*, *LibFuzzer*, *Neural Fuzzer*, *PyJFuzz*, *QuickFuzz*, *Wfuzz* [3].

## 7 Zaključak

Faz testiranje predstavlja vremenski efikasan način pronalaženja programskih grešaka te je stoga jako popularan. Cilj svakog pristupa faz testiranju je da ima što viši stepen automatizacije i da pokriva što veći deo testiranog koda, tj. da daje bolje rezultate [7]. Faz testiranje pokazuje prisustvo grešaka u aplikaciji ali ne može da garantuje otkrivanje grešaka u potpunosti. Upotrebom fuzzing tehnike osigurava se da je aplikacija robusna i sigurna, jer ova tehnika pomaže da se razotkrije većina uobičajenih ranjivosti [4].

## Literatura

- [1] CERT. Testiranje fuzz metodom. [https://www.cert.hr/wp-content/uploads/2019/04/NCERT-PUBDOC-2011-01-322\\_0.pdf](https://www.cert.hr/wp-content/uploads/2019/04/NCERT-PUBDOC-2011-01-322_0.pdf), 2011.
- [2] Matt Hillman. 15 minute guide to fuzzing. <https://www.f-secure.com/en/consulting/our-thinking/15-minute-guide-to-fuzzing>.
- [3] Computer Hope. Fuzz testing. <https://www.computerhope.com/jargon/f/fuzz-testing.htm>, nov 2019.
- [4] Priyanka Kothe. Fuzz testing(fuzzing) tutorial: What is, types, tools & example. <https://www.guru99.com/fuzz-testing.html>.
- [5] Kruna Matijević. Neke metode za automatizovano testiranje softvera. <http://elibrary.matf.bg.ac.rs/bitstream/handle/123456789/3375/masterKrunaMatijevic.pdf>, 2013.
- [6] Glenford J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.
- [7] Aleksandar Nikolić, Goran Sladić, Branko Milosavljević, and Zora Konjović. Detekcija sigurnosnih propusta faz testiranjem. *INFO M*, pages 9–14, jan 2019.
- [8] Marko Radenković. Heartbleed bag – nova/stara opasnost na internetu. <http://www.zastitapodataka.com/heartbleed-bag-novastara-opasnost-na-internetu/>, apr 2014.