CARNet
HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Testiranje Fuzz metodom

NCERT-PUBDOC-2011-01-322

Nacionalni CERT

Nacionalni CERT

Sadržaj

1 Uvod

Kako bi računalne aplikacije bile sigurne potrebno je testirati njihovu sigurnost. Jedna od metoda je nazvana testiranje Fuzz metodom. Tehniku je uspostavio Prof. Barton Miller sa sveučilišta u Wisconsinu 1988. kao studentski zadatak nazvan: "Operating System Utility Program Reliabilitiy – The Fuzz Generator". Osnovna ideja testiranja Fuzz metodom je davanje slučajnih ili pseudo-slučajnih podataka kao ulaz u testirani sustav. Ukoliko sustav prestane funkcionirati nakon nekog ulaza onda se taj niz podataka zabilježi za daljnju analizu. Danas se testiranje Fuzz metodom najviše koristi u velikim sustavima gdje se vrši testiranje na principu crne-kutije, zato što se pokazalo da Fuzz metoda daje jako dobar odnos cijene i vremena naspram kvalitete testiranja. Mogućnosti primjene su jako velike tako da se može testirati vrlo široki spektar različitih sustava od web-aplikacija, protokola, funkcija u kodu i sl. Neki od sigurnosnih propusta koji se mogu detektirati fuzzing metodama su:

- Preljevi spremnika,
- Umetanje SQL naredbi,
- Cross Site Scripting,
- Udaljeno izvršavanje naredbi i itd.

2 Fuzzer alati

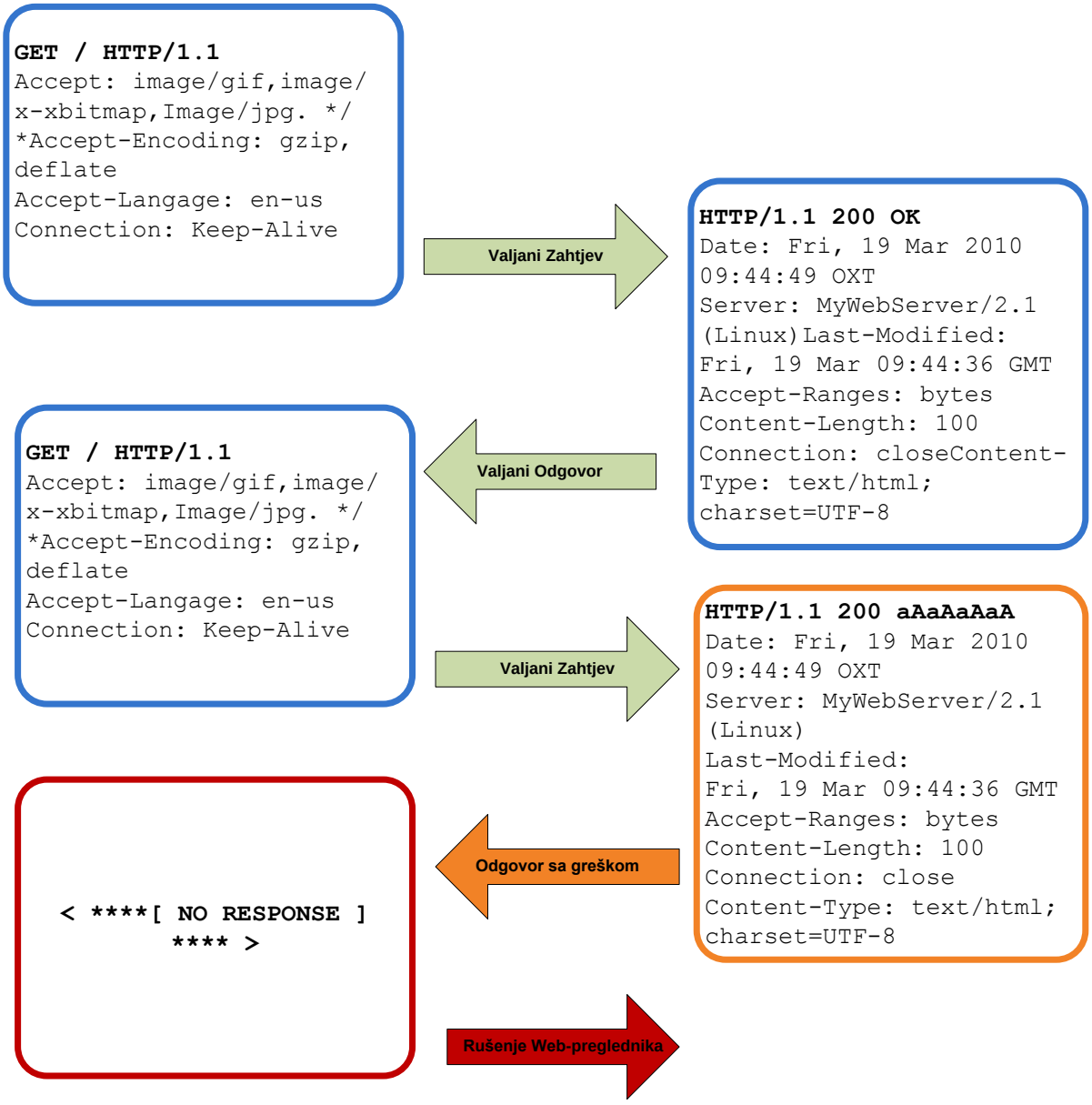
Za potrebe testiranja stvaraju se fuzzing alati koji mogu biti različitih tipova i namijenjeni različitim profilima korisnika. Budući da je fuzzing alat efikasniji ukoliko je prilagođeniji testnom slučaju, neki proizvođači prodaju biblioteke za razvoj fuzzing testova, no tada krajnji korisnici moraju sami implementirati konkretne testove što zahtjeva znanje i vrijeme. Prednost takvih testova što će su specijalizirani i samim time mogu podrobnije testirati sigurnosne propuste na testiranom objektu. Sa druge strane se nalaze gotovi potpuni alati koji uz namještanje određeni parametara omogućuju korisniku da vrši testiranje. Prednost ovakvog pristupa je jednostavnost, dok mana je manje kvalitetno testiranje u odnosu na testove koji se rade specifično za objekt koji se testira.
Druga podjela alata se vrši na temelju toga da li mogu pamtiti stanje ili ne. Alati koje ne pamte stanje su statički i ne mogu simulirati protokol. Kompleksniji alati su bazirani na modelu komunikacije i mogu vršiti komunikaciju sa poslužiteljem te po potrebi varirati parametre komunikacije, mijenjati pakete u komunikaciji i sl. Fuzzing alati koji su bazirani na modelu su se pokazali puno boljim u pronalaženju sigurnosnih propusta. Ovakvi napredniji testovi mogu doći do dubljih pogrešaka u sustavu.

3 Zero Day ranjivosti

Ranjivosti za koje se još ne zna i nastale su pri stvaranju softvera nazivaju se Zero Day ranjivosti. Izraz Zero Day je nastao u zajednici koja se bavi hakiranjem softvera i označava sigurnosne propuste za koji nitko drugi još ne zna ili barem proizvođač softvera još nije svjestan tog propusta. Tradicionalne provjere ranjivosti pokušavaju iskoristiti objavljenje i poznate sigurnosne propuste, njihova prednost je u tome što se provjera ranjivosti može vršiti i na sustavu koji korisnici trenutno koriste, bez bojazni da se naruši njegova dostupnost. Loša strana takvog pristupa je u tome što ne može pronaći nove sigurnosne propuste. Ovaj problem je najizraženiji na novim tehnologijama ili nadogradnjama nad postojećim tehnologijama, protokolima i sl. Za takve nove tehnologije u trenutku njihovog izlaženja ne postoji popis poznatih grešaka i propusta, pa zbog toga nije moguće raditi sigurnosne provjere uobičajenim softverom za provjeru ranjivosti. Testiranje Fuzzing alatima omogućuje pronalazak ranjivosti koji nisu do sada bile poznate. Sigurnosni stručnjaci preporučaju da bi testiranje trebalo vršiti periodički tokom razvoja softvera. Razlog tomu je cijena uklanjanja problema, koja manje ukoliko se pogreška uoči na vrijeme.

4 Testiranje Web Browsera Fuzzing alatima

Testiranje Fuzzing metodama može se također primijeniti na testiranje sigurnosti Web-preglednika. Moguće je testirati protokole poput HTTP-a i TLS-a te način na koji se preglednici nose sa pogreškama u XML zapisima. Većina preglednika je testirano da se dobro nosi sa dokumentiranim standardima, no ostaje pitanje kako se ponašaju kada trebaju primati i obraditi podatke koji nisu u skladu sa nekim od standarda. Testiranje se vrši tako da se u komunikaciju uvrštavaju po standardu loši ili nedozvoljeni podatci. Jedan takav slijed je prikazan na slici 1.



Slika 1. Fuzzing web-preglednika

Kao što je rečeno prije razlikujemo Fuzzing alate koje koriste statičke metode i one koji se baziraju na modelu. U slučaju testiranje web-preglednika u statičkom fuzzeru se na temelju analize blokova od kojih se sastoji HTTP zaglavlje može stvoriti algoritam koji će mijenjati pojedina polja koja neće biti u skladu sa standardima. Ukoliko je Fuzzing alat baziran na modelu, on mora sadržavati specifikaciju standarda ili protokola koji se ispituje.

4.1 Testiranje HTTP protokola

Osnova za prijenos informacija putem Interneta je protokol HTTP. Koristi ga se svakodnevno kako bi se pristupilo informacijama sa širokog spektra različitih uređaja, od računala, mobilnih uređaja i sl. Budući da je mogućnost pristupa Webu u bilo kojem trenutku postao vrlo bitan element svakodnevnog života potrebno je osigurati nesmetan pristup. HTTP se protokol koji ne pamti stanja i zasniva se zahtjev-odgovor komunikaciji. Svi Web-preglednici implementiraju HTTP protokol, a u načinu na koji to rade moguće je pronaći greške koje se mogu iskoristiti za dobivanje pristupa udaljenom računalu, rušenje samog Web-preglednika i sl. Treba napomenuti da TLS/SSL enkripcija ne zaštićuju ukoliko napadač cilja pogreške u radu HTTP protokola. HTTP promet se propušta kroz vatrozidove i u kombinaciji sa propustima u implementaciji na Web-preglednicima čini ozbiljni sigurnosni problem. Jedan od načina testiranja implementacije HTTP protokola u Web-pregledniku je testiranje Fuzz metodom. Pri testiranju na pogreške u HTTP zahtjeve odnosno odgovore se umeću slučajni parametri u različita polja HTTP zaglavlja te se zatim prati ponašanje Web-preglednika. Primjer jednog takvog zaglavlja je dan na slici 2.

```
000000  HTTP/1.1 200 OK\r\n
000011  Accept-Ranges: bytes\r\n
000027  ETag: "-1252324354"\r\n
00003c  Last-Modified: Tue, 16 Mar 2010 11:21:28 GMT\r\n
00006a  Content-Length: 321\r\n
00007f  Date: Tue, 16 Mar 2010 11:21:28 GMT\r\n
0000a4  Connection: close\r\n
0000b7  Server: HTTP Client Suite\r\n
0000d2  Content-Type: text/html\r\n
0000eb  Set-Cookie:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                        [not showing 32737 anomaly octets]
???116  !=!\r\n
???11b  \r\n
???11d  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"\r\n
???159     "http://www.w3.org/TR/html4/frameset.dtd">\r\n
???188  <HTML>\r\n
???190  <HEAD>\r\n
???198  <TITLE>A simple frameset document</TITLE>\r\n
???1c3  <META HTTP-EQUIV="refresh" content="1;URL=refresh.html">\r\n
???1fd  </HEAD>\r\n
???206  <FRAMESET>\r\n
???212        <FRAME src="index.html" NAME="target">\r\n
???240  </FRAMESET>\r\n
???24d  </HTML>   \r\n
???259     \r\n
```

Slika 2. HTTP odgovor sa anomalijom

Kada klijentska strana šalje zahtjev poslužitelju u zahtjevu daje informacije o sebi, jednako tako kada poslužitelj šalje odgovor šalje zaglavlja sa informacijama o sebi. Te informacije je moguće iskoristiti za bolji i usmjereniji napad na poslužitelja ili klijenta. Budući da tijelu HTTP poruke može biti bilo što, za testiranje robusnosti implementacije HTTP protokola pametnije je stvarati pogrešne zapise u zaglavljima i pratiti način na koji druga strana odgovara na njih.

4.2 Testiranje TLS protokola

Transport Layer Security (TLS) je kriptografski protokol koji omogućuje sigurnu komunikaciju putem Interneta. Koristi se u mnogo različitih primjene, a među njima su one koje se bave

Internet bankarstvom ili kupovanjem preko Interneta. Sigurnost u takvim primjenama je iznimno bitna te je ispravno funkcioniranje komunikacije putem TLS protokola od iznimne važnosti. Protokol TLS također koristi ASN.1 certifikate kako bi potvrdio autentičnost pružatelja usluga. Kompleksnost tih certifikata predstavlja ranjivost ukoliko je neki Web-preglednik loše implementirao standard. Do sada je već prijavljen i ispravljen znatan broj grešaka u implementaciji TLS protokola u različitim softverskim rješenjima.

TLS protokol se sastoji od puno koraka komunikacije te razmjena poruka između dva entiteta može biti dugačka. Budući da TLS podržava višestruke poruke broj kombinacija u komunikaciji je vrlo velik, iz toga za testiranje Fuzzing metodom ovog protokola potrebno je koristiti naprednije alate bazirane na modelu koji mogu dublje ispitati sigurnosne propuste. Fuzz aplikacija mora moći prepoznati sve poruke TLS implementacije i znati što je očekivani odgovor na njih. Primjer jedne takve pokvarene poruke je dan na slici 3.

```
000000      Server-Record
000000          content_type
000000              handshake_protocol                              . 16
000001          protocol-version
000001              SSL3                                        .. 03 00
000003          length                                          .. 00 7f
000005          Fragment
000005              iv                                              ()
000005              content
000005                          server-hello
000005                                  Handshake
000005            ...&.......... 02 00 00 26 03 00 00 00 00 00 00 00 00 00 00
000015            .............. 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                                                  [not showing 95 anomaly octets]
??????              mac                                           ()
??????              padding                                       ()
??????              pad_length                                    ()
```

Slika 3. Fuzzing TLS poruke

4.3 Testiranje XML zapisa

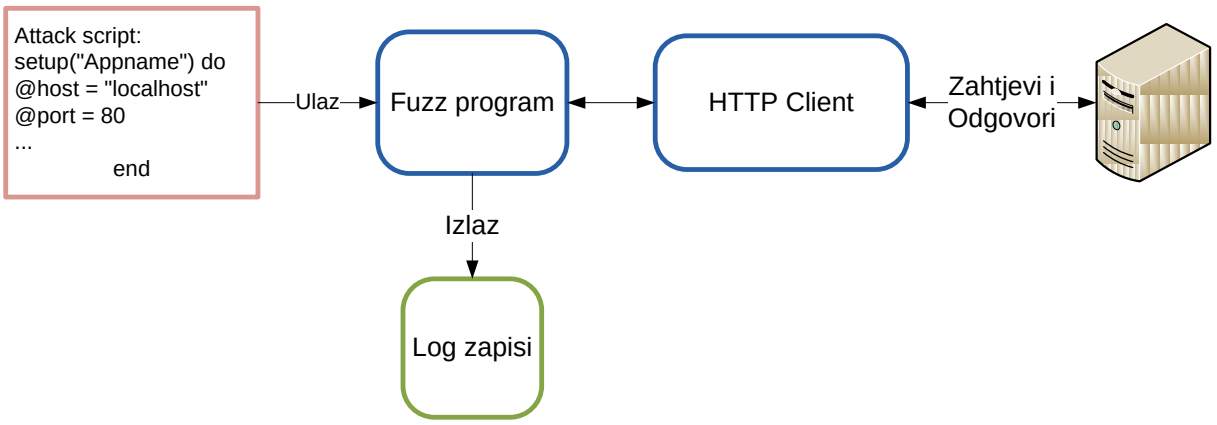
Popularni format za zapisivanje tekstualnih podataka, XML, se koristi u vrlo različitim poljima od vijesti (RSS feed) do zapisa različitih poslovnih podataka. XML format je vrlo svestran, no zbog toga može biti i vrlo kompleksan. U kompleksnosti se kriju potencijalne opasnosti pri obradi krivo zapisanih XML datoteka. Aplikacija koja koristi XML zapise se može podijeliti u tri djela:

- Parser
- Platforma
- Logika aplikacije

Parser i platforma omogućavaju da obradi XML zapis, a logika aplikacije dalje odlučuje što će sa tim podatcima. Ukoliko krivi podatci dođu do logike aplikacije može doći do neispravnog rada. Kako bi testirali obradu XML zapisa Fuzzing metodom mora se uzeti u obzir struktura sustava koji obrađuje XML zapise. Obrada zapisa ide kroz tri navedena djela i svaki se mora testirati posebno. XML parser će zamijetiti greške u strukturi XML zapisa, ali neće zamijetiti logički krive zapise u dobrom formatu. Fuzzing aplikacija mora varirati podatke koja bi mogla izazvati pogreške na sva tri nivoa. Interpretacija XML zapisa u Web-preglednicima može se testirati sa udaljene lokacije na kojoj je postavljen XMl zapis sa pogreškama.

5 Testiranje usluga Weba Fuzzing alatima

Slično kao i testiranje klijentskih Web-preglednika, fuzzing metodama moguće je testirati poslužiteljske aplikacije. Testiranje usluga Weba djelomično je slično testiranju Web-preglednika. Testira se način na koji usluga Weba barata sa dolazećim HTTP zahtjevima. Napadi na web usluge često kao početnu točku koriste forme za upis podataka. Kako bi pronašli potencijalne sigurnosne propuste Fuzzing alatima se napada upravo te forme i prati odgovore poslužitelja kako bi se dobile informacije o načinu kako obrađuje HTTP zahtjeve (Slika 4).



Slika 4. Arhitektura Fuzz alata

Napad se izvršava variranjem ulaznih podataka, kako bi se automatiziralo testiranje u što većoj mjeri moguće je napraviti skripte koje uz nekoliko pred definiranih parametara, poput adrese poslužitelja, mogu ispitivati sigurnost većeg broja usluga Weba. Primjer jedne takve skripte je dan na slici 5.

```
setup "Webapp" do
  @host = "10.0.0.2"
  @port = 3000
  @headers = "HTTP_ACCEPT_CHARSET" => "utf-8,*"
   attack "search-box" do
        many :get, "/search.php",
                :query => {:q => str(50)}
        many :get, "/search.php",
                :query => {:q => fix}
   end
   attack "post-page" do
        once :get, "/login.php", :query =>
                {:user => :admin, :pass => :admin}
        many :post, "/post.php", :query =>
                {:title => word, :body => byte(50)}
   end
end
```

Slika 5. Primjer skripte za fuzzing napad

Napredniji alati mogu sami pretraživati URL adrese za poljima koje korisnik može ispuniti i pokušati izvesti Fuzzing napad na njih. Nakon što je izveden napad da bi se došlo do važnih

zaključaka o sigurnosti i internoj strukturi Web-aplikacije potrebno je analizirati rezultate što često nije jednostavno. Kod Web-aplikacija se pokazalo kao najefikasnije koncentrirati na HTTP odgovore iz 500 kategorije, budući da oni predstavljaju interne greške na poslužitelju. Ukoliko poslužitelj vrati takav odgovor gotovo sigurno se može ustvrditi da je pronađena greška, u nekim slučajevima se u poruci o grešci nalazi i ispis stoga na poslužitelju koji omogućuje točniju analizu mjesta greške. Usporedbom vremenske oznake u odgovoru sa vremenskim oznaka poslanih zahtjeva lagano se može otkriti koji točno tip pokvarenog zahtjeva uzrokuje pogrešku.

6 Zaključak

Načina pronalaženja ranjivosti ima mnogo i niti jedna metoda nije idealna. U razvoju softvera svako testiranje košta i potrebno ga je učini što efikasnijim. Pogotovo u tom pogledu je vrlo uspješna Fuzzing metoda čiji se odnos vremena i broja nađenih pogrešaka pokazala vrlo dobrim. Fuzzing metoda nije idealna za sve vrste softvera ali može pokriti sasvim dovoljnu količinu različitih primjena, od web-aplikacija, pronalaženje Zero-Day ranjivosti u kodu ili testiranje robusnosti komunikacijskih protokola.

7 Literatura

1. Fuzz testing of web applications, Rune Hammersland i Einar Snekkenes, 2009
2. Automated Whitebox Fuzz Testing, Patrice Godefroid, Michael Y. Levin, David Molna, 2008
3. An Empirical Study of the Robustness of MacOS Applications
4. Using Random Testing, Barton P. Miller Gregory Cooksey Fredrick Moore, 2006
5. How to integrate
6. FUZZING and security testing into SDLC, Heikki Kortti, Jon Oltsik, Juan Rosales, 2008
7. Wireless Security:
8. Past, Present and Future, Sami Petäjäsoja, Tommi Mäkilä, Mikko Varpiola,
9. Miikka Saukko and Ari Takanen, 2008