Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/07BE12Budnic

OTKRIVANJE SIGURNOSNIH PROPUSTA U *SCADA* SISTEMIMA METODOM *FUZZ* TESTIRANJA

IDENTIFYING SECURITY VULNERABILITIES IN *SCADA* SYSTEMS VIA *FUZZ* TESTING METHOD

Spasoje Budnić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu opisani su problemi koji nastaju usljed propusta u razvoju programa. Objašnjena je fuzz metoda za pronalaženje sigurnosnih propusta. Fuzz metode se relativno često koriste u rasprostranjenim IT sistemima kao što su web sistemi, ali u slučaju SCADA sistema, gde postoje razni protokoli uključujući i vlasničke, ne postoji uniformno i dostupno rešenje za fuzz testiranje. S toga je cilj rada da istraži pristupe i predloži proširenje postojećih platformi tako da se omogući i SCADA fuzz testiranje.*

Ključne reči: *SCADA sigurnost, Modbus protokol, fuzz testiranje*

Abstract – *This paper describes the problems that arise from vulnerabilities in program development. The fuzz method for finding security leaks is explained. Fuzz methods are relatively commonly used in widespread IT systems such as web systems, but in the case of SCADA systems, where there are various protocols, including proprietary ones, there is no uniform and accessible solution for fuzz testing. Therefore, the aim of the paper is to explore approaches and propose extensions to existing platforms to enable SCADA fuzz testing.*

Keywords: *SCADA security, Modbus protocol, fuzz testing*

1. UVOD

Veliki broj sigurnosnih propusta u softveru nastaje zbog lošeg rukovanja ulazima ili zbog grešaka u logici programa. *SCADA* sistemi su postali izuzetno važni za većinu industrija širom svijeta jer kontrolišu kritične infrastrukture [1] kao što su električne mreže, vodovodi, gasovodi...

Stariji tipovi ovih sistema su koristili svoje izolovane mreže za komunikaciju [2]. Sistemi su uglavnom bili specifični po tome koju opremu i protokole koriste.

SCADA sistemi su izloženi internet prijetnjama, kao i ostali informacioni sistemi. Napadom na *SCADA* sisteme, postoji mogućnost da dođe do fizičkog uništenja stvari [1,3], što može biti zloupotrijebljeno od strane terorista. Mnogo je napada zabilježeno poslednjih nekoliko godina na industrijske sisteme.

Fuzz testiranje je tehnologija koja pokušava da otkrije sigurnosne ranjivosti slanjem nasumičnih ulaza aplikaciji ili uređaju [2, 4].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branislav Atlagić, docent.

2. *SCADA*

SCADA (Supervisory Control And Data Acquisition) podrazumijeva nadzor, kontrolu i prikupljanje podataka. Tradicionalni sistemi su se sastojali od jednog *PLC*-a koji je kontrolisao industrijsko postrojenje [5]. Vremenom se javila potreba za upravljanjem sistemima sa distribuiranim resursima pa je *SCADA* prerasla u distribuirani sistem. Danas su *SCADA* sistemi prisutni u skoro svim kritičnim infrastukturama [1]. *SCADA* sistem je zapravo jedan akviziciono-upravljački sistem. Akvizicija podataka podrazumijeva dobavljanje vrijednosti sa digitalnih ili analognih mjerila iz polja. Akvizicija je neizostavan proces u svim *SCADA* sistemima jer se sve odluke donose na osnovu prikupljenih i obrađenih podataka [5].

Kada su ovi sistemi prvobitno razvijani, vrlo malo pažnje se posvećivalo zaštiti od zlonamernih napada. Sistemi su bili fizički izolovani, koristili su vlasničku opremu, kao i specifične komunikacione protokole. Iz dobro poznatih razloga, ovakvi sistemi su sada pogodni za mrežne napade. Nedostatak autentifikacije pošiljaoca, korišćenje komercijalnih operativnih sistema, upotreba podrazumijevanih lozinki ili odsustvo lozinki za pristup uređajima samo su neki od razloga zašto ovi sistemi mogu biti laka meta [2].

3. INDUSTRIJSKI PROTOKOLI

Industrijski protokoli predstavljaju grupu komunikacionih protokola specifičnih za *SCADA* sisteme. Tradicionalno, industrijski protokoli su često vezani za specifičnu oblast primjene, još češće za proizvođača opreme [5]. U Americi je dominantan *DNP3* protokol, dok su u Evropi *IEC 60870-5-101* i *IEC 6087-5-104* [6]. U starijim sistemima još uvijek se koriste *Modbus, Fieldbus,* kao i mnogi drugi vlasnički protokoli [6].

4. MODBUS

Modbus protokol razvijen je 1979. godine od strane *Modicon* kompanije. Protokol i specifikacija su potpuno otvoreni što je u velikoj mjeri doprinijelo njegovoj popularnosti [2, 5].

Modbus-ASCII i *Modbus-RTU* [5] su dvije prvobitne implementacije protokola za rad preko serijskog *UART* kanala, tekstualna i binarna verzija. S pojavom *ethernet*-a razvijaju se verzije *Modbus-TCP* [2] i *Modbus-UDP* koje predstavljaju enkapsulaciju *Modbus-RTU* protokola preko *TCP* i *UDP* protokola.

Tabela 1. *Struktura Modbus poruke*

	MBAP				*PDU*	
	Transaction ID	*Protocol ID*	*Length*	*Unit ID*	*Function Code*	*Payload*
Veličina u oktetima	2	2	1	1	1	N

U tabeli 1 se vidi struktura *Modbus-TCP* poruke. Poruka se sastoji od zaglavlja (*MBAP)* i tijela (*PDU).*

Transaction ID predstavlja identifikator transakcije. Zahtjev i odgovor treba da imaju identičan identifikator transakcije. *Master* stanica generiše identifikator po nekom internom algoritmu, dok *slave* stanica treba da vrati odgovor sa istim identifikatorom.

Protocol ID je identifikator protokola. U ovom slučaju je uvijek 0.

Length označava broj okteta koji slijede poslije njega.

Unit ID je identifikator stanice, koristan je kada se više *slave* stanica nalazi iza *hub*-a. *Hub* prosljeđuje svima poruku, pa je identifikator stanice jedini način da se odredi kome poruka pripada.

Function Code je kôd funkcije koju udaljena stanica treba da izvrši. Veličine je jednog okteta pa je broj mogućih vrijednosti od 0 do 255, međutim u slučaju zahtjeva validne vijednosti su od 0 do 127 [2]. Vrijednosti od 128 do 255 koriste se ukoliko zadata funkcija nije uspješno izvršena. Tada udaljena stanica vraća odgovor gdje je *FunctionCode* zahtjeva uvećan za 0x80.

Modbus uređaji i simulatori ne moraju da podržavaju sve funkcije definisane protokolom, međutim većina njih bi trebalo da podržava funkcije za manipulisanje digitalnim i analognim registrima (tabela 2).

Tabela 2. *Osnovne Modbus poruke*

Function code	*Modbus message*
1	*Read Coils*
2	*Read Discrete Inputs*
3	*Read Holding Registers*
4	*Read Input Registers*
5	*Write Single Coil*
6	*Write Single Register*

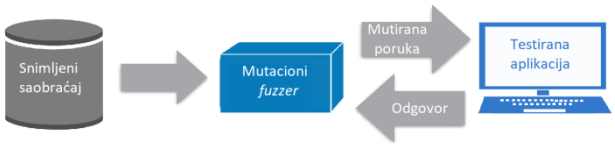
5. *FUZZ* TESTIRANJE

Fuzz testiranje je sigurnosna tehnika gdje se kreiraju ulazi koji se proslijeđuju ciljanom softveru sa namjerom da se pronađu nedostaci ciljanog programa. Najčešći nedostatak koji se uočava *fuzz* testiranjem je loša validacija ulaznih podataka. Nerijetko *fuzz* testiranje može da otkrije i nedostatke u aplikacionoj logici. Danas većina modernih programa ima dobru provjeru ulaza i može da obradi slučajno izabrane nizove okteta bez rušenja. Stoga se moderni alati za *fuzz* testiranje prilagođavaju modernim programima kako bi bili što efikasniji. Moderni alati biraju niz okteta na osnovu određene semantike. Obično postoji više slojeva aplikacije kroz koje podaci moraju proći prije nego što dođu do same aplikacione logike [4].

5.1 Vrste *fuzzer*-a

Na osnovu kreiranih testnih podataka, *fuzzer*-i se dijele u dvije grupe: generacioni i mutacioni.

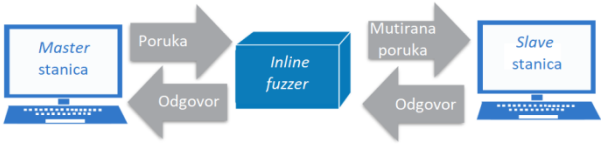
Generacioni *fuzzer*-i generišu testne ulaze na osnovu pravila kako treba da izgleda pravilan ulaz [3]. Najjednostavniji generacioni *fuzzer*-i kreiraju ulaze sa slučajnim odabirom.



Slika 1. *Mutacioni fuzzer*

Mutacioni *fuzzer*-i (slika 1) funkcionišu tako što čitaju poznate ulaze u koje ubacuju *loše* oktete i na taj način kreiraju testne ulaze [3]. Većina mutacionih *fuzzer*-a koristi prethodno snimljeni saobraćaj kao polaznu osnovu za mutiranje ulaza. *GPF (General Purpose Fuzzer)* [4] ne zahtjeva poznavanje protokola ili strukture podataka. Koristi jednostavne heuristike da pogodi granice i mutira ulaze na osnovu njegovih pretpostavki. Stoga *fuzzer*-i ne moraju da poznaju strukturu ulaza koje *napadaju*. Mogu slijepo da čitaju *dobre* ulaze i da mutiraju bez poznavanja semantike protokola [4].

Nedostatak ovakvih *fuzzer*-a je što su u stanju da testiraju serversku stranu, dok klijenti ostaju netestirani. Nakon kreiranja paketa, šalju ih na specificiranu adresu i port. Klijenti ne prihvataju saobraćaj kao serveri pa samo alati koji prate mrežni saobraćaj mogu da testiraju klijente.



Slika 2. *Inline fuzzer*

Inline fuzzer-ima (slika 2) se nazivaju alati koji prate mrežni saobraćaj. Koriste tehniku presretanja *Man in the Middle* kako bi presreli mrežne podatke. U kritičnim infrastrukturama veliki broj protokola radi sa vremenski osjetljivim sesijama pa se ne mogu testirati sa alatima koji zahtijevaju veliko skladištenje paketa, već se koriste *inline* alati [4].

5.2 Testiranje vlasničkih protokola

Vlasnički protokoli koje koristi *SCADA* oprema najčešće nisu dobro razumljivi. Domenski eksperti najčešće nemaju dovoljno vremena ili sposobnosti da izvrše testiranje nad protokolima koje oprema koristi [4].

S druge strane, eksperti koji se bave *fuzz* testiranjem mogu obaviti posao umjesto domenskih eksperata. Problem nastaje u kritičnim infrastrukturama gdje su domenski eksperti veoma oprezni i ne dozvoljavaju strancima da rade sa njihovom opremom [4].

LZFuzz se koristi za testiranje vlasničkih *SCADA* protokola gdje je gramatika nepoznata. Pronalazi tokene poruka koristeći *Lempel-Ziv*-ov algoritam. *LZFuzz* koristi informacije iz tokena kako bi mutirao poruke [7].

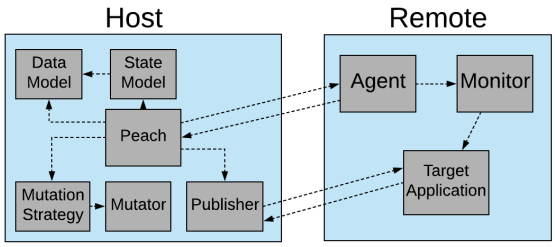
Međutim, ovo je *fuzz* tehnika zasnovana na mutaciji, pa je teško primijeniti na protokol sa komplikovanom ulaznom gramatikom. Stoga većina alata za *fuzz* testiranje *SCADA* protokola trenutno koristi gramatiku protokola [3].

6. *PEACH FUZZER*

Peach je platforma za automatsko testiranje i prevenciju napada pronalaženjem propusta u sistemima [8]. Koristi tehniku bijele kutije za testiranje programa. *Peach Community Edition* je softver otvorenog koda koji je korišten za testiranje *Modbus* protokola u ovom radu.

Peach odvaja model podataka od samog sistema koji se testira, kao i od samog alata. Ovaj alat pruža sistem za praćenje i detektovanje grešaka. Alat je veoma fleksibilan i proširiv. Ukoliko je potrebno testirati neki specifičan sistem, moguće je dodati nove komponente kako bi se postigla veća efikasnost.

Peach pruža mogućnost kreiranja različitih *fuzzing* strategija, kao i kreiranje specifičnih mutatora podataka. Kombinacija ovoga omogućava potpunu kontrolu nad testiranjem. Jednostavna proširivost alata omogućava testerima da urade različita testiranja, a da pri tom ne prave novi alat.



Slika 3. *Arhitektura Peach platforme*

Na slici 3 prikazana je arhitektura *Peach fuzzer*-a. Većina modernih *fuzzer*-a teži sličnoj arhitekturi koja omogućava brzu i laku adaptaciju. Slika 3 prikazuje najčešći slučaj gdje se testirana aplikacija nalazi na udaljenom računaru. Prilikom testiranja aplikacija opisanih u ovom radu, *fuzzer* i testirana aplikacija se nalaze na istom računaru.

Model definiše model podataka koji se testira. U ovom slučaju je definisan *Modbus* model podataka. Najviše vremena treba posvetiti definisanju modela podataka. U zavisnosti od toga kako je modelovan protokol, zavisi i uspješnost samog testiranja.

Izdavač (*Publisher*) predstavlja ulazno-izlazni *interface*. Izdavači pružaju stvarni transport i realizaciju. Primjer ugrađenih izdavača su *TCPPublisher*, *UDPPublisher*, *ConsolePublisher*... U ovom radu testiran je *Modbus* protokol koji se prenosi preko *TCP* protokola, pa je moguće koristiti *TCPPublisher*. Za *Modbus-RTU* protokol neophodno je kreirati novog izdavača.

Strategija testiranja je logika koja predstavlja na koji način će se modifikovati elementi modela podataka. Strategija određuje koji mutatori će se koristiti i na koji način, kao i da li će se modifikovati neki dio podataka ili ne. Sekvencijalna i nasumična su neke od ugrađenih strategija.

Mutator se koristi za kreiranje podataka. Mutatori najčešće koriste postojeće zadate vrijednosti koje mutiraju. Mutatori mogu kreirati nove elemente, kreiraju niz zadate dužine sa slučajno izabranim elementima.

Agenti su posebni procesi koji se mogu pokrenuti lokalno ili na udaljenom računaru i pokreću jedan ili više monitora. Zadatak monitora je praćenje nekog procesa.

6.1 *Peach Pit*

Peach Pit file je *XML file* koji sadrži neophodne informacije za testiranje. *Peach Pit* je konfiguracioni dokument koji sadrži informacije o modelu podataka koji se testira, kao i o agentima i monitorima koji se koriste prilikom testiranja.

6.2 *Modbus* model podataka

Model bi trebalo da predstavlja validnu gramatiku protokola. *Peach fuzzer* koristi tehniku bijele kutije baziranu na gramatici protokola, nastojeći da pronađe propuste u programu.

Osnovni cilj prilikom testiranja aplikacije je pogoditi granične slučajeve i pri tom smanjiti broj testnih slučajeva koji će odmah biti odbačeni. Klasičan primjer su protokoli koji sadrže sigurnosnu provjeru, kao što su *CRC* ili *LRC*. Ukoliko ova provjera nije validna, vjerovatnoća da testni podaci ne budu odbačeni već prilikom prve validacije je minimalna. U slučaju *Modbus-TCP* protokola sigurnosna provjera se nalazi u sklopu *TCP* protokola.

Da bi testirali logiku aplikacije, podaci moraju biti *skoro* validni, odnosno neophodno je da prođu ulazne validacije i dopru do same logike.

Jedan od načina kreiranja podataka je da se u svakom testnom slučaju šalje validno zaglavlje. Prilikom kreiranja modela treba voditi računa da uređaji i sistemi ne moraju da podržavaju sve protokol funkcije, pa bi kreiranje testnih slučajeva za funkcije koje nisu podržane smanjilo efikasnost pronalaženja propusta.

U ovom radu objašnjena su polja funkcija iz tabele 2. Prilikom testiranja aplikacija, pored ovih, testirane su i funkcije *WriteMultipleCoils* i *WriteMultipleRegisters.*

U ovom primjeru mutirana su samo polja koja se nalaze poslije koda funkcije u *Modbus* poruci: *Start Address*, *Quantity*, *Output Address* i *Value.* Sva polja su mutirana tako da im se postavljaju najmanja i najveća vrijednost kao i nasumično izabrane vijednosti. Takodje korišteni su mutatori koji dupliraju elemente ili prave nizove elemenata.

Testiranjem graničnih vrijednosti i dupliranjem elemenata se mogu izazvati prelivanje *buffer*-a i aritmetičko prelivanje. Prelivanje *buffer*-a se dešava prilikom zapisivanja podataka van granica *buffer*-a. Može se iskoristiti da se zamijeni povratna adresa na *stack*-u i pokrene izvršavanje zlonamernog koda. Aritmetičko prelivanje nastaje kada rezultat aritmetičike operacije prelazi granice promjenjive u koju treba da sačuva rezultat.

6.2.1 *Start Address* polje

Start Address polje predstavlja adresu prvog registra u funkcijama za dobavljanje vrijednosti. Funkcije koje sadrže ovo polje su *Read Coils*, *Read Discrete Inputs*, *Read Holding Registers* i *Read Input Registes*.

6.2.2 *Quantity* polje

Quantity polje u funkcijama za dobavljanje vrijednosti označava koliko registara je potrebno dobaviti. Nalazi se poslije *Start Address* polja. Funkcije koje sadrže ovo polje su *Read Coils, Read Discrete Inputs, Read Holding Registers, Read Input Registers, Write Multiple Coils* i *Write Multiple Registers.*

6.2.3 *Output Address* polje

Output Address polje označava adresu registra u koji je potrebno upisati vrijednost. Funkcije koje sadrže ovo polje su *Write Singe Coil i Write Single Register*.

6.2.4 Value polje

Value polje označava vrijednost koju je potrebno upisati u registar. Funkcije koje sadrže ovo polje su: *Write Singe Coil* i *Write Single Register.*

7. PRIMJENA NAD REALNIM APLIKACIJAMA

Testirane su sledeće *Modbus* aplikacije otvorenog koda: *MOD_RSSIM, PyModSlave* i *EasyModbusTCP* server simulator. Na sve testirane aplikacije primjenjen je model podataka opisan u poglavlju 6.2. Korištene su sekvencijalna i nasumična strategija sa najviše 10000 iteracija. Uočene su određene anomalije u ponašanju aplikacija.

7.1 *MOD_RSSIM*

MOD_RSSIM je simulator *SCADA* udaljene stanice. Prilikom testiranja ovog simulatora uočeno je da poslije određenog vremena simulator prestaje da odgovara na zahtjeve. S obzirom da je simulator testiran u iteracijama, na početku svake iteracije se uspostavlja veza sa simulatorom, a na kraju iteracije prekida. U ovom slučaju desio se *DoS (Denial of Service)* napad. Jedan od slučajeva kada se desi *DoS* je slanje samog zaglavlja u iteracijama.

7.2 *PyModSlave*

Prilikom testiranja *PyModSlave* simulatora uočeno je da već na početku testiranja simulator postaje nedostupan za akcije sa korisničkog interfejsa. Simulator komunicira sa *fuzzer*-om, ali nije moguće izvršiti nijednu komandu putem korisničkog interfejsa.

7.3 *EasyModbusTCP* server simulator

Prilikom testiranja *EasyModbusTCP* server simulatora, *windows debugger* je registrovao prestanak rada aplikacije veći broj puta. Ukoliko monitor, u ovom slučaju *windows debugger*, registruje nevalidno stanje, *fuzzer* pokušava da reprodukuje to stanje ponovo, kako bi verifikovao sekvencu koja je to stanje izazvala.

Jedna od grešaka je aritmetičko prelivanje kada je kôd funkcije veći od 127. Naknadnom forenzikom je utvrđeno da nema provjere da li je kôd funkcije validan. Prilikom postavljanja greške da funkcija nije podržana dolazi do aritmetičkog prelivanja.

Poruka *WriteMultipleCoils* može *srušiti* simulator ukoliko je *Quantity* veći od stvarnog broja poslatih registara.

6.1 Verifikacija uspjeha

Ukoliko se prilikom testiranja *SCADA* sistema uoči bilo kakva anomalija u ponašanju, može se smatrati da je testiranje bilo uspješno. Ukoliko se prilikom testiranja ne dokaže nijedan propust u implementaciji, to nije dokaz da propusti ne postoje. Možda samo testni slučajevi nisu dobro kreirani.

Ovim radom je ustanovljeno da u izvornom kodu programa opisanog u poglavlju 7.3 treba dodati provjeru kako bi se spriječilo aritmetičko prelivanje. Takođe treba dodati i provjeru da li prilikom *WriteMultipleCoils* zahtjeva primljeni broj vrijednosti digitalnih registara odgovara vrijednosti polja *Quantity.*

8. ZAKLJUČAK

Osnovna zamisao ovoga rada je ukazivanje na sigurnosne propuste u *SCADA* sistemima nastale lošim rukovanjem ulazima, kao i propusti unutar same logike.

Metod *fuzz* testiranja postaje sve popularniji zbog brzine u pronalaženju grešaka. Moderni alati koriste gramatiku za kreiranje ulaza i postižu efikasnije rezultate od alata koji nasumično biraju ulaze.

Za realizaciju rada bilo je neophodno poznavanje *Modbus* industrijskog protokola, kao i upoznavanje sa tehnikom *fuzz* testiranja.

Za sprovođenje testiranja izabran je *Peach Community Edition* alat otvorenog koda, koji se relativno lako modifikuje i konfiguriše da radi sa *Modbus* protokolom.

U ovom radu predstavljeno je testiranje *Modbus* serverskih aplikacija, a unaprijeđenje alata bi podrazumijevalo testiranje i klijentske strane.

8. LITERATURA

[1] Kyle Coffey, Richard Smith, Leandros Maglaras and Helge Janicke, *Vulnerability Analysis of Network Scanning on SCADA Systems*, 2018

[2] Mehdi Sabraoui, Jeffery L. Hieb, and James H. Graham, *Protocol Fuzzing for Cyber Security and Hardening of Industrial Control Systems*, 2014

[3] Hyunguk Yoo, Taeshik Shon, *Grammar-based Adaptive Fuzzing: Evaluation on SCADA Modbus Protocol*, 2016

[4] Rebecca Shapiro, Sergey Bratus, Edmond Rogers, Sean Smith, *Do it yourself SCADA vulnerability testing with lzfuzz*, 2011

[5] Branislav Atlagić, *Softver sa kritičnim odzivom, projektovanje SCADA sistema*, 2015

[6] Frances Cleveland, *IEC TC57 Security Standards for the Power System's Information Infrastructure – Beyond Simple Encription*, 2006

[7] Sergey Bratus, Axel Hansen, Anna Shubina, *LZfuzz: a fast compression-based fuzzer for poorly documented protocols*, 2008

[8] Peach alat *(https://www.peach.tech/products/peach-fuzzer/*) (pristupljeno u septembru 2019)

Kratka biografija:

Spasoje Budnić rođen je 1995. godine u Trebinju. Završio je Srednju Tehničku Školu u Trebinju 2014. godine. Fakultet tehničih nauka u Novom Sadu upisao je 2014. godine. Diplomski rad pod nazivom *Akvizicija podataka u SCADA sistemima upotrebom Series 5 protokola* odbranio je 2018. godine.