Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/20BE42Trifunovic

PRIMENA TEHNIKA OBFUSKACIJE IZVORNOG KODA NAD PROGRAMSKIM JEZIKOM *SOLIDITY*

APPLICATION OF SOURCE CODE OBFUSCATION TECHNIQUES IN THE *SOLIDITY* PROGRAMMING LANGUAGE

Petar Trifunović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu će biti predstavljene mogućnosti primene tehnika obfuskcaije nad izvornim kodom napisanom u Solidity programskom jeziku. Rad uključuje i opis implementacije ovih tehnika izvršene u programskom jeziku Go.*

Ključne reči: *obfuskacija, neprozirni iskazi, blokčejn, pametni ugovori*

Abstract – *This work will present the possibilities of applying obfuscation techniques over source code written in the Solidity programming language. The work also includes a description of the implementation of these techniques written in the Go programming language.*

Keywords: *obfuscation, opaque predicates, blockchain, smart contracts*

1. UVOD

Metode obfuskacije[1] primenjuju se nad kodovima pisanim u mnogim programskim jezicima u svrhu povećanja bezbednosti i skrivanja semantike koda. Stariji i popularniji jezici su u ovom smislu u prednosti u odnosu na novije, s obzirom na to da je više vremena utrošeno na njihovu detaljnu analizu. Dodatno, jezici u čijem procesu prevođenja postoji jasno definisana varijanta međukoda pružaju dodatne naprednije mogućnosti obfuskacije, pa je vremenom za takve jezike stvoreno mnoštvo obfuskacionih alata. Dobri primeri su međukodovi *.NET* i *Java* programskih jezika (*Common Intermediate Language* za *.NET*, *Java Bytecode* za *Java*-u).

Sa druge strane, blokčejn tehnologije se još uvek mogu smatrati relativno mladim, kao i programski jezici specijalizovani za pisanje pametnih ugovora. Za ove jezike uglavnom ne postoje razvijeni alati za obfuskaciju. Uz to, specijalizovani jezici za pametne ugovore odlika su javnih blokčejnova sa učlanjivanjem bez dozvole [2] (*Ethereum, Avalanche, Cardano...*), a ono što karakteriše ovakve tipove blokčejn tehnologija jeste nepoverenje između učesnika. U takvom sistemu, svako prikrivanje detalja, pa i obfuskacija koda, može samo povećati nepoverenje, tako da se potpuno dostupan izvorni kod u neizmenjenoj formi može smatrati za nešto poželjno, ako ne i očekivano. Ipak, nepostojanje alata za povećanje tajnosti semantike koda može otežati testiranje. Iako je moguće pisati i testirati pametne ugovore bez da se sam izvorni kod učini javno dostupnim u bilo kom trenutku, ovakvo nešto zahteva dodatni napor programera i korišćenje nestandardnih tehnika (uspostavljanje privatne varijante blokčejn mreže, javno baratanje isključivo kodovima niskog nivoa...). Zbog toga postoji smislen razlog za ulaganje napora za kreiranje obfuskacionih alata za jezike za rad sa blokčejnom. Takođe, dobar obfuskator bi mogao da se iskoristi da deo, ili čitavu implementaciju logike neke blokčejn aplikacije, čije su funkcionalnosti javno dostupne, sakrije od njenih korisnika iz bilo kog razloga[3].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dušan Gajić, vanr. prof.

[1] *Zamagljivanje* u bukvalnom prevodu sa engleskog jezika

[2] U nastavku teksta će se termini *blokčejn* i *blokčejn tehnologije* odnositi upravo na javne blokčejnove sa učlanjivanjem bez dozvole

2. FORMATIRANJE I PISANJE TEKSTA

Blokčejn je posebna vrsta distribuiranog sistema. Tačnije, blokčejn predstavlja jednu od mogućih implementacija tehnologije po imenu distribuirana glavna knjiga (eng. *Distributed Ledger Technology*, u nastavku *DLT*). Pametni ugovori su odlika blokčejn tehnologija i povećavaju mogućnost upotrebe istih. U nastavku je dat pregled pojmova najvažnijih za razumevanje blokčejna.

2.1. Distribuirani sistemi

Distribuirani sistem predstavlja skup nezavisnih računarskih elemenata koji korisniku deluju kao jedan koherentan sistem [1]. Često, distribuirani sistem podrazumeva i fizičku distribuiranost procesa učesnika, tako da se oni izvršavaju na geografski udaljenim mašinama.

2.2. Distribuirana glavna knjiga

Distribuirana glavna knjiga predstavlja jedan tip distribuiranog sistema specifičan po tome što pretpostavlja da nema bilo kakvog vida poverenja između učesnika. DLT je vid distribuirane baze podataka čiju kopiju poseduje svaki učesnik sistema i koja se sinhronizovano ažurira uspostavljanjem konsenzusa bez učešća centralizovanog arbitra [2]. Koncept sistema u

[3] U ovom slučaju, korisnici bi morali da imaju potpuno poverenje u rad aplikacije, za šta su pogodnije privatne blokčejn varijante.

kome više zasebnih učesnika mora da radi sinhronizovano uz određenu vrstu dogovora uveli su prvi put Lamport et al. [3] i dali osnovu za tehnologiju kakva je DLT.

2.3. Blokčejn

DLT predstavlja samo apstraktan koncept i zahteva konkretizaciju kako bi se dobio funkcionalan sistem. Jedna takva konkretna implementacija distribuirane glavne knjige jeste blokčejn. 1991. i 1993. godine prvi put se pojavljuju ideje o kreiranju blokova podataka međusobno povezanih u lanac korišćenjem tehnika kriptografije [4, 5]. Godine 2008. anonimni autor pod pseudonimom Satoši Nakamoto objavljuje predlog funkcionalne blokčejn mreže – *Bitcoin* [6], a 2009. objavljuje implementaciju na osnovu predloga i pušta zvaničnu *Bitcoin* mrežu u upotrebu.

Blokčejn sistem predstavlja implementaciju DLT-a u kojoj se stanje sistema beleži u vidu kriptografski povezanih blokova. Svaki blok sastoji se od skupa transakcija od kojih svaka predstavlja atomičnu promenu stanja sistema. Učesnici u mreži određenim protokolom postižu konsenzus oko toga koji blok treba sledeći uključiti u lanac. Kada se blok prihvati i bude uključen u lanac sve transakcije u njemu postaju trajno zabeležene u sistemu, jer se njihovo izbacivaje iz sistema ne može izvršiti bez ogromnog i neisplativog utroška resursa.

Ovo je važna karakteristika blokčejna – učesnici nemaju poverenja jedni u druge, ali imaju poverenja u ispravan rad sistema. Oni moraju da nešto ulože (računarske resurse, „novac“ u vidu važeće blokčejn valute...) kako bi se izvršilo ažuriranje stanja. Ukoliko njihov predlog ažuriranja nije zlonameran i bude prihvaćen od strane drugih članova mreže, biće nagrađeni zbog činjenice da su nešto uložili kako bi stanje sistema napredovalo. Ako je ažuriranje zlonamerno, ogromne su šanse da će im ulog propasti, tako da nema smisla ni pokušavati ovako nešto. Ovakav princip rada ohrabruje učesnike da imaju poverenja u rad blokčejna. Opisane karakteristike blokčejna predstavljaju sažetak opisa *Bitcoin*-a iz rada Satošija Nakamota [6], ali su prihvaćene i implementirane i u mnogim blokčejn sistemima nastalim nakon *Bitcoin*-a.

3. PAMETNI UGOVORI

Koncept pametnih ugovora uveo je Nik Sabo u članku iz 1996. godine [7]. Pametni ugovor jeste naprednija verzija standardnih ugovora koji su se donedavno pretežno svodili na potpisivanje papira na kome je izlistano šta uključene strane obećavaju da će ispuniti. Takvi standardni ugovori zahtevaju i učešće još jednog člana koji najčešće ima zakonsku moć da primora potpisnike ugovora na ispunjenje obećanja, kao i moć da ih kazni u slučaju neispunjenja istih. Pametni ugovor je računarski podržan transakcioni protokol koji izvršava uslove iz ugovora; njihov cilj jeste da osigura ispunjavanje obećanja iz ugovora, da svede na miminum odstupanja od ugovora, bila ona namerna ili slučajna, kao i da smanji potrebu za učešćem centralizovanog posrednika [7].

Sa pojavom blokčejna pametni ugovori dobijaju izuzetno važnu praktičnu primenu. Blokčejn kao koncept u potpunosti ima smila i bez pametnih ugovora, ali bez njih bi bio ograničen na rad sa transakcijama vrlo jednostavne logike. Na primer, *Bitcoin* je prvenstveno namenjen za razmenu sredstava izraženih u *BTC*-u, odnosno u *Bitcoin*-ovoj kriptovaluti [8]. Bez mogućnosti pisanja pametnih ugovora[4], ne bi bilo moguće definisati uslove koje primalac mora da ispuni da bi dokazao da su poslata sredstva namenjena baš njemu, pa bi bilo koji učesnik mogao da „presretne“ i ukrade poslatu sumu. *Ethereum* je blokčejn sistem koji predstavlja svojevrsnu platformu za kreiranje distribuiranih aplikacija napisanih pomoću pametnih ugovora, tako da u *Ethereum*-u i drugim kompatibilnim blokčejnovima pametni ugovori igraju ogromnu ulogu i predstavljaju srž sistema.

3.1. Jezici za pisanje pametnih ugovora

Pametni ugovori u blokčejn sistemima nude mogućnost definisanja proizvoljne logike koja se treba izvršiti u mreži kako bi uslovi ugovora bili ispunjeni. Ovakva definicija praktično poistovećuje pametan ugovor sa kodom računarskog programa, s obzirom na to da i programski kod predstavlja specifikaciju koraka koje računar treba da izvrši kako bi sproveo u delo neki algoritam. Zbog toga se i logika pametnog ugovora predstavlja putem koda napisanog u nekom programskom jeziku. Od 2009. godine pojavili su se razni jezici namenjeni za pisanje pametnih ugovora za konkretne blokčejn sisteme.

Svaki čvor blokčejna učestvuje u validaciji predloženih transakcija. U procesu validacije čvorovi izvršavaju kod pametnog ugovora vezanog za konkretnu trasakciju. Pametan ugovor u ovakvom sistemu pisan u kodu nekog od standardnih programskih jezika koji koriste memorijske i procesorske resurse samog čvora bio bi pogodno tlo za razne vrste napada, kao što su narušavanje memorijskog prostora računara, upošljavanje procesora predugim ili beskonačnim petljama itd. Blokčejn sistemi se protiv ovih problema bore na sledeće načine:

a) **Korišćenje jezika koji nisu Tjuring-kompletni.** Programski jezik je Tjuring-kompletan ukoliko sve što se može na bilo koji način izračunati može biti izračunato i putem koda pisanog u tom jeziku [9]. Ovakav jezik bi iz pomenutih razloga bio problem u blokčejn sistemu. Zato se mogu koristiti posebni jezici koji nisu Tjuring-kompletni za pisanje pametnih ugovora. Ovakav pristup smanjuje opasnosti, ali ograničava mogućnosti pametnih ugovora.

b) **Izvršavanje koda pametnih ugovora u specijalnim virtuelnim okruženjima.** Ovakav pristup povećava bezbednost s obzirom na to da izoluje okruženje u kome se izvršava pametan ugovor od ostatka mašine, nalik na virtuelnu mašinu. Virtuelno okruženje bi moglo da spreči narušavanje memorije i procesora čvora, ali i dalje postoje konstrukcije koje bi narušile rad blokčejn sistema. Na primer, beskonačna petlja bi „ukočila“ virtuelno okruženje.

[4] Mogućnosti pametnih ugovora u *Bitcoin*-u su veoma jednostavne, dok je njihova prava moć zaživela sa pojavom *Ethereum* blokčejna.

c) **Naplata izvršavanja pametnog ugovora.** Ovako je moguće sprečiti učesnike od pisanja ugovora koji zahtevaju previše procesorkse moći, jer bi cena izvršavanja bila prevelika.

Bitcoin Script je jezik kreiran za pisanje jednostavnih pametnih ugovora za *Bitcoin* blokčejn i od navedenih koristi tehniku a). Ovaj jezik podržava pisanje jednostavne logike u pametnim ugovorima, ali postoje pokušaji unapređenja i proširenja njegovih mogućnosti.

Ethereum [10] je blokčejn sistem koji predstavlja platformu za razvoj decentralizovanih aplikacija pomoću pametnih ugovora. Termin pametnog ugovora se danas pretežno i vezuje za *Ethereum*, (u predlogu rada *Bitcoin*-a ovaj termin nije ni pomenut). *Ethereum* koristi navedene tehnike b) i c). Kod koji se piše u bilo kom specijalizovanom jeziku višeg nivoa biva preveden u kod *Ethereum* virtuelne mašine (eng. *Ethereum Virtual Machine Code*, u nastavku *EVM* kod) [10]. Ona obezbeđuje da izvršavanje ne zađe u kontekst i memoriju računara na kome je *EVM* kod pokrenut. Ovaj kod niskog nivoa je Tjuring-kompletan, pa se protiv problema kao što su beskonačne petlje bori korišćenjem koncepta gasa. Za svaku transakciju neophodno je da pošiljalac navede koliko maksimalno je spreman da plati izvršenje (izraženo u *ETH*, kriptovaluti *Ethereum* blokčejna). To je suma koju učesnik mreže mora da priloži uz svaku transakciju. Ta suma naziva se gas. Ukoliko transakcija zahteva manju ili jednaku količinu gasa od priložene, moći će da se izvrši. Ukoliko zahteva više, transakcija će biti prekinuta. Ovo sprečava kočenje mreže beskonačnim petljama, jer bi takva petlja zahevala beskonačno mnogo gasa [10].

4. *SOLIDITY* PROGRAMSKI JEZIK

Solidity je objektno orijentisan jezik visokog nivoa namenjen za implementiranje pametnih ugovora [11]. Idejni tvorac jezika je Gavin Vud. Inspirisan je jezicima *C++*, *Python* i *JavaScript* [11]. *Solidity* ne uvodi previše sintakse koja nije već poznata od ranije u drugim jezicima. Osnovni entitet u *Solidity* jeziku jeste ugovor (eng. *contract*), što je koncept nalik na klasu u standardnim objektno orijentisanim jezicima – ima atribute, metode, modifikatore pristupa. Dokumentacija nudi i opis sintakse i leksičkih pravila ovog jezika [11]. *Solidity* se koristi za razvoj pre svega na na *Ethereum*-u, ali i na drugim kompatiblnim blokčejnovima.

5. METODE OBFUSKACIJE PROGRAMSKOG KODA

Obfuskacija koda predstavlja način na koji se programski kod prevodi iz originalne u (logički) ekvivalentnu varijantu, ali manje razumljivu [12]. Šta je *manje razumljivo* određeno je time nad kojom varijantom koda se obfuskacija vrši. Ukoliko se radi o izvornom kodu, kod je manje razumljiv ukoliko je ljudskom oku teže protumačiti ga. Ako se vrši obfuskacija međukoda ili asemblerskog koda, potrebno je da kod bude manje razumljiv alatima za obrnuti inženjering (eng. *reverse engineering*), odnosno da bude teže rekreirati reprezentaciju koda u jezicima visokog nivoa.

Ovaj rad bavi se obfuskacijom izvornog koda. U nastavku je dat pregled tehnika obfuskacije iskorišćenih u predloženom rešenju. Većina ovih tehnika je pobrojana i opisana i u. [12].

5.1. Promena imena promenljivih

Nazivi promenljivih često nose mnogo informacija o značenju pojedinih delova koda. Zbog toga je dobro izmeniti ove nazive u kodu. Ovo se može uraditi na nekoliko načina – pretvoriti imena u nasumične nizove karaktera; pretvoriti imena u konkretne, ali besmislene nizove karaktera; izvršiti konkretan algoritam nad postojećim imenima (heširanje) kako bi se dobila nova.

5.2. Brisanje komentara

Komentari mogu sadržati značajne informacije o semantici koda, pa ih je uvek dobro eliminisati.

5.3. Izmena numeričkih konstanti

Kod postaje vizuelno mnogo kompleksiniji ukoliko se numeričke vrednosti zamene aritmetičkim izrazima. Ipak, jednostavne aritmetičke izraze je kompajler u stanju da u toku parsiranja izvornog koda prevede u numeričku konstantu ekvivalentnu tom izrazu. *Jednostavnim* se mogu smatrati izrazi pisani u jednoj liniji, odnosno u jednoj instrukciji izvornog koda, koji koriste trivijalne aritmetičke operacije. Na primer, izraz (1) bio bi optimizovan od strane kompajlera i aritmetički izraz bio bi preveden u konstantu 12. Razlaganjem ove dodele na više instrukcija optimizacija bi se mogla zaobići.

$$a = 4 * 3 \qquad (1)$$

5.4. Direktno umetanje tela funkcije

Direktno umetanje tela funkcije na mesto njenog poziva (eng. *function inlining*) čini kod dužim, vizuelno kompleksnijim i poništava inicijalnu namenu izdvajanja koda u funkcije. Umetanje tela podrazumeva detekciju poziva funkcija, zamenu parametarskih promenljivih u telu funkcije konkretnim argumentima i rešavanje eventualnih konflikata u nazivima promenljivih.

5.5. Neprozirni iskazi

Neprozirni iskazi (eng. *opaque predicates*) prvi put su pomenuti u radu Kolberga et al. [13]. Iskazi u programskom jeziku se odnose na izraze čija je rezultujuća vrednost *boolean* tipa, odnosno može imati vrednost *true* i *false*.

Vrednost neprozirnih iskaza je poznata onome ko vrši obfuskaciju koda, ali treba da onome ko pokušava da rastumači bude teško da zaključi koju vrednost će iskaz imati pri izvršavanju. Iskazi mogu biti nezavisni od konteksta izvršenja i da pri svakom pokretanju programa imaju istu vrednost, ili da budu zavisni od konteksta. Iskazi zavisni od konteksta pogodni su za kreiranje dinamičkih neprozirnih iskaza, čija se vrednost ne može zaključiti pre pokretanja programa. Za funkcionisanje ovakvih iskaza potrebno je izvršiti kombinaciju nekoliko njih. Princip rada dinamičkih iskaza detaljnije je objašnjen u [14].

6. REŠENJE

Rešenje predloženo u ovom radu predstavlja implementaciju obfuskacionih metoda navedenih u prethodnom poglavlju nad pametnim ugovorima pisanim u *Solidity* programskom jeziku. Implementacija rešenja data je u programskom jeziku *Go*. U svrhu implementacije korišćen je zvaničan *Solidity* kompajler [11]. On pruža mogućnost generisanja apstraktnog sintaksnog stabla odnosno *JSON* fajla koji pruža važne informacije o elementima kompajliranog ugovora. Rešenje analizira ovaj fajl i izvlači detalje relevantne za primenu pomenutih obfuskacionih metoda, kao što su imena promenljivih i imena funkcija, njihove pozicije u fajlu izvornog koda, detalji o izvršenim pozivima funkcija, argumenti prosleđeni funkcijama itd. Za izvlačenje dodatnih informacija korišćeni su regularni izrazi. Samo rešenje predstavljeno je u vidu konzolne *Go* aplikacije koja neizostavno uključuje obfuskaciju imena promenljivih, komentara i numeričkih konstanti. Na korisniku aplikacije je izbor između umetanja tela funkcija na mesto njihovog poziva, i neprozorinih iskaza.

Manipulacija kodom vrši se u više faza i struktura fajla izvornog koda se menja, za razliku od *JSON* fajla. Stoga je uvedena struktura podataka koja predstavlja kombinaciju crveno-crnog stabla i dvostruko spregnute lančane liste. Namenjena je za praćenje promena nastalih u izvornom kodu. Ona omogućava da se brzom pretragom dobije informacija o tome gde se određena linija koda nalazi u izmenjenom izvornom kodu, a omogućava i brzo ažuriranje radi beleženja novonastalih izmena.

7. ZAKLJUČAK

S obzirom na sličnost sa standardnim objektno orijentisanim jezicima, činjenica da je *Solidity* jezik specifične namene ne unosi mnogo neočekivanih komplikacija u pokušaje manipulacije njegovim kodom. Ipak, koliko god zadržavala logiku originala, nijedna manipulacija izvornim kodom nije besplatna. Gotovo je sigurno da će *Solidity* kod nad kojim se primene tehnike obfuskacije zahtevati veću količinu gasa nego original. Ovaj rad pokazuje da je moguće izmeniti kod i time njegovu semantiku sakriti do određene mere. Svakako, treba razmotriti da li je cena obfuskacije, bilo da je ona finansijska ili se ogleda u potrošnji resursa računara, manja od „zarade“ dobijene tim izmenama.

8. LITERATURA

[1] M. Van Steen, A.S. Tanenbaum, „A brief introduction to distributed systems“, *Computing* 98, pp. 967-1009, 2016.

[2] R. Ugarte, J. Luis, „Distributed Ledger Technology (DLT): Introduction“, Banco de Espana Article 19/18, October 2018.

[3] L. Lamport, R. Shostak, M.Pease, „The Byzantine Generals Problem“, *Transactions on Programming Languages and Systems*, volume 4, issue 3, July 1982.

[4] S. Haber, W.S. Stornetta, „How to Time-stamp a Digital Document“, *Journal of Cryptography*, volume 3, pp. 99-111, 1991.

[5] D. Bayer, W.S. Stornetta, S. Haber, „Improving the Efficiency and Reliability of Digital Time-Stamping“ u *Sequences II: Methods in Communication, Security and Computer Science*, New York: Springer-Verlag, 1993, pp. 329-334

[6] S. Nakamoto, „Bitcoin: A peer-to-peer electronic cash system“, https://bitcoin.org/bitcoin.pdf , 2008.

[7] N. Szabo, „Smart Contracts: Building Blocks for Digital Markets“, 1996. [Online]. Dostupno na: **https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html** (pristupljeno u septembru 2022.)

[8] J. Frankenfield, „What is Cryptocurrency?“, May 2022. [Online]. Dostupno na: **https://www.investopedia.com/terms/c/cryptocurrency.asp** (pristupljeno u septembru 2022.)

[9] S. Kepser, „A Simple Proof of the Turing-completeness of XSLT and XQuery“, u *Proc. Extreme Markup Languages*, Quebec, 2004.

[10] V. Buterin, „Ethereum Whitepaper“, 2013. [Online]. Dostupno na: **https://ethereum.org/en/whitepaper/** (pristupljeno u septembru 2022.)

[11] [Online]. Dostupno na: **https://docs.soliditylang.org/en/v0.8.17/** (pristupljeno u septembru 2022.)

[12] C. Collberg, C. Thomborson, D. Low, „A Taxonomy of Obfuscating Transformations“, 1997.

[13] C. Collberg, C. Thomborson, D. Low, „Manufacturing cheap, resilient, and stealthy opaque constructs“, *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '98*, 1998.

[14] D. Xu, J. Ming, D. Wu, „Generalized Dynamic Opaque Predicates: A New Control Flow Obfuscation Method“, *Lecture Notes in Computer Science*, Springer, pp. 323-342

Kratka biografija:

Petar Trifunović rođen je u Nišu 1998. god. Diplomski rad na Elektronskom fakultetu u Nišu iz oblasti Elektrotehnike i računarstva – Računarstvo i informatika odbranio je 2021. god.

kontakt: peki.trifunovic@gmail.com