MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică
Departamentul Ingineria Software și Automatică

Admis la susţinere
Şef departament:
Fiodorov Ion, conf. univ., dr.
_____
„__” _____ 2021

Analiza și identificarea unui framework de testarea a interacțiunii cu Smart Contracts pe Blockchain

Teză de master

Student: _____ **Lungu Mihail, gr. TI-201M**

Coordonator: _____ **Nastasenco Veaceslav, conf. univ., dr.**

Consultant: _____ **Cojocaru Svetlana, lect. univ.**

Chişinău, 2022

Rezumat

Teza de master prezentată de studentul Lungu Mihail își propune să detecteze cele mai bune tehnici și abordări de testare pentru Smart Contracts implementate pe un Blockchain, iar ca exemplu de mediu, toate contractele au fost implementate pe Ethereum Blockchain testnet.

Chainlink oferă utilizatorilor un set întreg de garanții pentru a asigura un mecanism oracol extrem de sigur și de încredere. Prin construirea acestor caracteristici cheie pe Chainlink, contractele inteligente pe orice blockchain pot acum accesa date din afara lanțului fără a-și sacrifica valoarea de bază a determinismului, oferind o bază solidă de la care să construiască viitorul automatizării bazate pe date.

Scopul lucrării constă în analiza și identificarea celei mai bune soluție pentru testarea automată a rețelei descentralizate de noduri care furnizează date și informații din surse off-blockchain la Smart Contracts în blockchain prin oracole.

Noile tehnologii necesită noi tipuri de testare. După cum am văzut în exemplul Blockchain și al oracolelor, metodele tradiționale de testare nu sunt suficiente. Trebuie să se meargă cu un nivel mai sus și să se testeze aplicațiile la nivel de arhitectură.

Pentru realizarea scopului au fost întreprinse:

1. Utilizarea celui mai bun limbaj pentru interactiunea cu Blockchain și Kuberenetes, care este Golang (GO).
2. S-a creat o modalitate de a implementa medii efemere care să permită testarea diferitor arhitecturi și moduri de a implementa întregul pachet de aplicații care modelează medii de producție în execuție

Chainlink a rezolvat ceea ce era cunoscut sub numele de „problema oracolului". Problema oracolului provine dintr-o problemă cu Smart Contracts pe rețelele blockchain și modul în care acestea sunt complet izolate de lumea exterioară. Smart Contracts își obțin de obicei datele externe de la „Oracles" (puncte de date, API-uri) - și aici este problema. Contractele inteligente sunt doar la fel de „inteligente" ca și informațiile furnizate de oracole. Dacă un contract inteligent este furnizat cu cod rău intenționat sau date inexacte, contractul îl va procesa oricum, deoarece este doar cod - și ceea ce iese ar fi imprevizibil, greșit sau mai rău.

Teza de master a fost elaborate în 3 capitole, cu descriere succintă a fiecăruia:

Capitolul 1 - descrie domeniul căruia îi aparține sistemul și prezintă o cercetare a acestuia în vederea rezolvării problemei identificate;

Capitolul 2 - descrie abordarea potrivită pentru testarea unui Smart Contract, aplicind metoda DefectChecker

Capitolul 3 - descrie utilizarea aplicației și documentația sistemului

Abstract

The master's thesis presented in this report by the student Lungu Mihail aims to detect best testing techniques and approaches for Smart Contracts deployed on a Blockchain. As an environment example all contracts were deployed on Ethereum Blockchain testnet.

These are just some of the many features offered by Chainlink that provide users with a whole set of guarantees to ensure a highly secure and reliable oracle mechanism. By building out these key features on Chainlink, smart contracts on any blockchain can now access off-chain data without sacrificing its core value of determinism, providing a solid foundation from which to build out the future of data-driven automation.

The scope of the thesis work is to analyse and find out best solution for automated testing of decentralized network of nodes that provide data and information from off-blockchain sources to on-blockchain smart contracts via oracles.

New technologies require new types of testing. As we have seen in the example of Blockchain and oracles traditional testing methods are not enough. We have to go one level higher and test applications at architecture level. To achieve this we have done following things:

1. Use the best language for Blockchain and Kuberenetes which is Golang (GO)
2. We have created a way to deploy ephemeral environments which allows us to test different architectures and ways to deploy the whole stack of applications which model running production environments

Chainlink has solved what was known as the "oracle problem". The oracle problem originates from an issue with smart contracts on blockchain networks and how they are completely isolated from the outside world. The smart contracts typically obtain their external data from "Oracles" (data points, APIs) – and this is where the problem lies. Smart contracts are only as "smart" as the information delivered to them by the oracles. If a smart contract is provided with malicious code or inaccurate data, the contract will still process it anyway because it is just code – and what comes out would be unpredictable, wrong, or worse.

The structure of this work is organized by chapters. Below they will be listed along with a short description:

Chapter 1 – describes the field to which the system belongs and presents a research of it in order to solve the identified problem;

Chapter 2 – describes the right approach for testing an Smart Contract, which is DefectChecker approach

Chapter 3 – describes application usage and system documentation

Contents

Introduction

On the blockchain, smart contracts are Turing-complete programs. Even when bugs are discovered, they are immutable and cannot be changed. As a result, it's critical to make sure smart contracts are bug-free and well-designed before publishing them to the blockchain. A contract defect is a flaw, error, or flaw in a smart contract that leads it to deliver an inaccurate or unexpected result or behave in unanticipated ways.

Detecting and eliminating contract faults can help you avoid errors and make your applications more reliable. Decentralized cryptocurrencies have gained a lot of attention in recent years. Decentralized cryptocurrencies use the blockchain concept as their underlying technology to ensure that these systems are scalable and secure without the need for centralized governance.

Smart contracts have several characteristics that make them appealing to hackers. On the one hand, many smart contracts store valuable Ethers and are unable to conceal their balance, providing a financial incentive for hackers to attack.

Smart contracts, on the other hand, operate on a permission-less network, which implies that hackers can easily inspect all transactions and bytecode in order to uncover weaknesses in the contracts. Even when faults are discovered, smart contracts cannot be updated. As a result, it's critical to make sure smart contracts are bug-free and well-designed before putting them on Ethereum.

The financial services business is another area where smart contracts are appropriate. For example, the technology might be used to automate trade clearing and settlement, bond coupon payment, and even insurance claim computation and payout.

Smart contracts, while their apparent uses in finance, are adaptable enough to be used in virtually any business where dollars, digital assets, or any type of digital information must be transferred between parties. The equipment leasing industry, for example, has made substantial use of these contracts in the real world to improve lease agreements.

References

1. Bitcoin contract. URL: https://en.bitcoin.it/wiki/Contract (Date: 2019-01-30).
2. Solidity-example-crowdfunding. URL: https://github.com/zupzup/solidity-example-crowdfunding (Date: 2021-09-30).
3. D. Macrinici, C. Cartofeanu, and S. Gao. Smart contract applications within blockchain technology: A systematic mapping study. Telematics and Informatics, vol. 35, no. 8, 2018, pp. 2337–2354.
4. C. D. Clack, V. A. Bakshi, and L. Braine. Smart contract templates: foundations, design landscape and research directions. CoRR, vol. abs/1608.00771, 2016.
5. A 50 million hack just showed that the dao was all too human. URL: https://www.wired.com/2016/06/50- million-hack-just-showed-dao-human/ (Date: 2021-09-30).
6. D. McAdams. An ontology for smart contracts. URL: https://cryptochainuni.com/wpcontent/uploads/Darryl-McAdams-An-Ontology-for SmartContracts.pdf (Date: 2021-10-07).
7. N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts sok. In Proc. Of the 6th International Conference on Principles of Security and Trust, 2017, pp. 164–186.
8. G. Destefanis, A. Bracciali, R. Hierons, M. Marchesi, M. Ortu, and R. Tonelli. Smart contracts vulnerabilities: A call for blockchain software engineering. ResearchGate, 2018.
9. Safer smart contracts through type-driven development. URL: https://publications.lib.chalmers.se/records/fulltext/234939/234939.pdf (Date: 2021-09-30).
10. Bitcoin script. URL: https://en.bitcoin.it/wiki/Script (Date: 2021-09-30)
11. R. O'Connor. Simplicity: A new language for blockchains. CoRR, vol. abs/1711.03028, 2017.
12. Flint. URL: https://github.com/flintlang/flint (Date: 2021-09-30).
13. Vyper. URL: https://github.com/ethereum/vyper (Date: 2021-09-29).
14. T. Kasampalis, D. Guth, B. Moore, T. Serbanuta, V. Serbanuta, D. Filaretti, G. Rosu, and R. Johnson. Iele: An intermediate-level blockchain language designed and implemented using formal semantics. University of Illinois, Tech. Rep., http://hdl.handle.net/2142/100320, July 2018.
15. I. Sergey, A. Kumar, and A. Hobor. Scilla: a smart contract intermediate-level language. CoRR, vol. abs/1801.00687, 2018.
16. Plutus core specification. URL: https://github.com/input-output-hk/plutus/tree/master/plutus-core-spec (Date: 2021-09-30).

17. D. Harz and W. J. Knottenbelt. Towards safer smart contracts: A survey of languages and verification methods. CoRR, vol. abs/1809.09805, 2018.
18. P. L. Seijas, S. Thompson, and D. McAdams. Scripting smart contracts for distributed ledger technology. Cryptology ePrint Archive, Report 2016/1156, 2016, https://eprint.iacr.org/2016/1156.
19. Contract. URL: https://en.bitcoin.it/wiki/Contract (Date: 2021-09-30).
20. Ethereum contract security techniques and tips. URL: https://github.com/ethereum/wiki/wiki/Safety (Date: 2021-09-29).
21. E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G.Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A.Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proc. of the Thirteenth EuroSys Conference, 2018, pp.30:1–30:15.
22. K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun. Potential risks of hyperledger fabric smart contracts. In Proc. of the 2019
23. IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2019, pp. 1–10. 300m in cryptocurrency accidentally lost forever due to bug. URL: https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether (Date: 2021-09-30).
24. Smart contract weakness classification and test cases. URL: https://smartcontractsecurity.github.io/SWCregistry/ (Date: 2021-09-29).
25. Decentralized application security project. URL: https://dasp.co (Date: 2021-09-29).
26. Security considerations. URL: https://solidity.readthedocs.io/en/latest/security-considerations.html (Date: 2021-09-29).
27. Vulnerabilities description. URL: https://github.com/trailofbits/slither/wiki/Vulnerabilities-Description (Date: 2021-09-30).
28. Smart contract weakness classification and test cases. URL: https://smartcontractsecurity.github.io/SWCregistry/ (Date: 2021-10-22).
29. N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts sok. In Proc. Of the 6th International Conference on Principles of Security and Trust, 2017, pp. 164–186.
30. Ether — the crypto-fuel for the ethereum network. URL: https://www.ethereum.org/ether (Date: 2021-09-30).

31. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254–269.
32. D. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. URL: https://ethereum.github.io/yellowpaper/paper.pdf (Date: 2021-11-30).
33. Michelson language. URL: https://www.michelson-lang.com/ (Date: 2021-11-30).
34. B. C. Pierce. Types and Programming Languages, 1st ed. The MIT Press, 2002.
35. Solidity. URL: https://github.com/ethereum/solidity
36. Liquidity. URL: https://github.com/OCamlPro/liquidity
37. Grigore Roșu and T. F. Șerbănuță. An overview of the k semantic framework. The Journal of Logic and Algebraic Programming, vol. 79, no. 6, 2010, pp. 397–434.
38. S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov. Smartcheck: Static analysis of ethereum smart contracts. In Proc. of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, 2018, pp. 9–16.
39. S. Kalra, S. Goel, M. Dhawan, and S. Sharma. Zeus: Analyzing safety of smart contracts. In Proc. of the Network and Distributed Systems Security (NDSS) Symposium, 2018.
40. K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A.Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin. Formal verification of smart contracts: Short paper. In Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security, 2016, pp. 91–96.
41. P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev. Securify: Practical security analysis of smart contracts. In Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 67–82.
42. E. Hildenbrandt, M. Saxena, X. Zhu, N. Rodrigues, P. Daian, D. Guth, B. Moore, Y. Zhang, D. Park, A. Stefanescu, and G. Rosu. Kevm: A complete semantics of the ethereum virtual machine. In Proc. of the 2018 IEEE 31st Computer Security Foundations Symposium, 2018, pp. 204–217.
43. Mythril. URL: https://github.com/ConsenSys/mythril-classic
44. L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz. Vandal: A scalable security analysis framework for smart contracts. CoRR, vol. abs/1809.03981, 2018
45. Rattle. URL: https://github.com/trailofbits/rattle
46. Manticore. URL: https://github.com/trailofbits/manticore

47. L. G. Meredith and M. Radestock. A reflective higher-order calculus. Electronic Notes in Theoretical Computer Science, vol. 141, 2005, pp. 49–67.
48. Bitcoin weaknesses. URL: https://en.bitcoin.it/wiki/Weaknesses
49. A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, Enabling blockchain innovations with pegged sidechains. 2014. https://blockstream.com/sidechains.pdf
50. Mediawiki.URL: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki
51. Simplicity. URL: https://github.com/ElementsProject/simplicity
52. Grishchenko I., Maffei M., Schneidewind C. A semantic framework for the security analysis of ethereum smart contracts – technical report (2018). URL: https://secpriv.tuwien.ac.at/tools/ethsemantics.
53. Formalization of ethereum virtual machine in lem. URL: https://github.com/pirapira/eth-isabelle
54. Ewasm: design overview and specification. URL: https://github.com/ewasm/design
55. Michelson: the language of smart contracts in tezos. URL: http://www.liquiditylang.org/doc/reference/michelson.html
56. Why michelson? URL: https://www.michelson-lang.com/why-michelson.html
57. Plutus core semantics. URL: https://github.com/kframework/plutus-core-semantics
58. Plutus implementation and tools. URL: https://github.com/input-output-hk/plutus
59. The extended utxo model. URL: https://github.com/input-output-hk/plutus/tree/master/docs/extendedutxo
60. Is it smart to use smart contracts? URL: https://plutusfest.io/presentations/Philip-Wadler/Wadler30.pdf
61. Solidityx. URL: https://solidityx.org/
62. Bamboo. URL: https://github.com/pirapira/bamboo
63. Logikon. URL: https://github.com/logikon-lang/logikon
64. Ivy: Bitcoin smart contracts. URL: https://github.com/ivy-lang/ivy-bitcoin (Date: 2021-10-22).
65. Çagdas Bozman, M. Iguernlala, M. Laporte, F. L. Fessant, and A. Mebsout. Liquidity: Ocaml pour la blockchain. Journées Francophones des Langages Applicatifs 2018, 2018.
66. Yul. URL: https://solidity.readthedocs.io/en/latest/yul.html (Date: 2021-10-22).
67. Rchain and rholang. URL: https://www.rchain.coop/platform (Date: 2021-10-22).
68. D. Ancona, V. Bono, and M. Bravetti. Behavioral Types in Programming Languages. Hanover, MA, USA: Now Publishers Inc., 2016.

69. G. Wood. LLL. URL: https://lll-docs.readthedocs.io/en/latest/index.html (Date: 2019-01-30).
70. Upgradable contract with solidity. URL: https://gist.github.com/Arachnid/4ca9da48d51e23e5cfe0f0e14dd6318f (Date: 2021-10-22).
71. Proxy libraries in solidity. URL: https://blog.zeppelin.solutions/proxy-libraries-in-solidity-79fbe4b970fd (Date: 2021-10-22).
72. The pact smart-contract language. URL: http://kadena.io/docs/Kadena-PactWhitepaper.pdf (Date: 2021-10-22).
73. T. Chen, X. Li, Y. Wang, J. Chen, Z. Li, X. Luo, M. H. Au, and X. Zhang. An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks. CoRR, vol. abs/1712.06438, 2017.
74. E. Albert, P. Gordillo, A. Rubio, and I. Sergey. GASTAP: A gas analyzer for smart contracts. CoRR, vol. abs/1811.10403, 2018.
75. M. Marescotti, M. Blicha, A. E. J. Hyvärinen, S. Asadi, and N. Sharygina. Computing exact worst-case gas consumption for smart contracts. In Proc. of the International Symposium on Leveraging Applications of Formal Methods, 2018.
76. J. Hoffmann, A. Das, and S. Weng. Towards automatic resource bound analysis for ocaml. CoRR, vol. abs/1611.00692, 2016.
77. J. Baeten. A brief history of process algebra. Theoretical Computer Science, vol. 335, no. 2, 2005, pp.131–146.
78. H. Deyoung and F. Pfenning. Reasoning about the consequences of authorization policies in a linear epistemic logic. In Proc. of the Workshop on Foundations of Computer Security, 2009.
79. S. Thompson and P. L. Seijas. Marlowe: Financial contracts on blockchain. Lecture Notes in Computer Science, vol. 11247, 2018, pp. 356–375.
80. G. Bigi, A. Bracciali, G. Meacci, and E. Tuosto. Validation of decentralized smart contracts through game theory and formal methods. Lecture Notes in Computer Science, vol. 9465, 2015, pp. 142–161.
81. M. Bartoletti and R. Zunino. Bitml: A calculus for bitcoin smart contracts. In Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 83–100.
82. Y. Hirai. Defining the ethereum virtual machine for interactive theorem provers. Lecture Notes in Computer Science, vol. 10323, 2017, pp. 520–535.
83. J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defining Smart Contract Defects on Ethereum," IEEE Transactions on Software Engineering, 2020.
84. Geth. [Online]. Available: https://github.com/ethereum/go-ethereum

85. T. Chen, Y. Zhang, Z. Li, X. Luo, T. Wang, R. Cao, X. Xiao, and X. Zhang, "TokenScope: Automatically Detecting Inconsistent Behaviors of currency Tokens in Ethereum," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1503–1520.
86. T. Chen, Y. Feng, Z. Li, H. Zhou, X. Luo, X. Li, X. Xiao, J. Chen, and X. Zhang, "Gaschecker: Scalable analysis for discovering gasinefficient smart contracts," IEEE Transactions on Emerging Topics in Computing, 2020.