Analiza napadov na pametne pogodbe osnovane na okoljih Ethereum virtualnega stroja

Gal Gantar, Matevž Pesek
Univerza v Ljubljani, Fakulteta za Računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
gg5490@student.uni-lj.si, matevz.pesek@fri.uni-lj.si

Izvleček

V zadnjem desetletju je zanimanje za kriptovalute močno naraslo. S tem je prišlo tudi do porasta kapitala, zaklenjenega v pametnih pogodbah, kar pa je hkrati povečalo tveganje za napade. Za preprečevanje napadov na pametne pogodbe je potrebna analiza programerskih vzorcev in okolja, v katerem se pametne pogodbe izvajajo. V prispevku obravnavamo varnost omrežja Ethereum in njegovo vlogo v razvoju decentraliziranih aplikacij (angl. decentralised Applications – dApps). Pregledamo uporabo Ethereumovega Virtualnega stroja (EVM), programskega jezika Solidity in zbirnika pri gradnji pametnih pogodb. Poleg tega opišemo varnostne izzive, s katerimi se sooča Ethereumov ekosistem, in analiziramo tri pretekle napade na decentralizirane platforme. Prispevek prinaša tudi konkreten primer napada na pametno pogodbo, ki izkorišča ranljivost v njeni implementaciji. Rezultat prispevka je prikaz pametne pogodbe, ki ob standardni uporabi deluje pravilno, vendar vsebuje ranljivost, ki jo je mogoče izkoristiti za neavtorizirano spreminjanje njenega stanja.

Ključne besede: binarni napadi, Ethereum, kibernetska varnost, Solidity

Analysis of smart contract exploits in Ethereum virtual machine environment

Abstract

Over the past decade, interest in cryptocurrencies has grown significantly. As a result, there has been an increase in the capital locked in smart contracts, which has simultaneously increased the risk of attacks. Preventing attacks on smart contracts requires an analysis of coding patterns and the environment in which smart contracts operate. This article examines the security of the Ethereum network and its role in the development of decentralized applications (dApps). We review the use of the Ethereum Virtual Machine (EVM), the Solidity programming language, and the compiler in the construction of smart contracts. Additionally, we describe the security challenges faced by the Ethereum ecosystem and analyze three past attacks on decentralized protocols. This article also presents a concrete example of an attack on a smart contract exploiting a vulnerability in its implementation. The results of this article are the demonstration of a smart contract that functions correctly under normal use, but contains a vulnerability that can be exploited to alter its state without authorization.

Keywords: binary exploit, cyber security, Ethereum, Solidity

1 UVOD

Kriptovaluta **Ethereum** je prinesla revolucijo v načinu, kako razumemo in uporabljamo tehnologijo veriženja blokov. Ethereum, ki je bil ustanovljen leta 2015, je zasnovan kot odprtokodna platforma, ki omogoča razvijalcem, da zgradijo in izvajajo decentralizirane aplikacije. Ethereuma je platforma, na kateri lahko vsak uporabnik neodvisno od centralnih avtoritet razvija in izvaja decentralizirane aplikacije ter pametne pogodbe. To omogoča in spodbuja inovacije na področju decentraliziranih finančnih storitev, digitalnih identitet ter drugih aplikacij [1, 2].

Osnova Ethereumovega ekosistema je **Ethereumov virtualni stroj** (angl. Ethereum Virtual Machine – EVM). EVM je virtualno okolje, ki omogoča izvajanje posebnih programov oziroma pametnih pogodb, ki so napisani v notranjem jeziku Ethereuma, imenovanem **Solidity**. Solidity je visokonivojski jezik, ki je bil posebej zasnovan za pisanje pametnih pogodb in je zelo podoben JavaScriptu, vendar pa zaradi cene računanja na Ethereumovih pametnih pogodbah postaja vse pogostejša praksa prepisovanje delov pametnih pogodb v Solidity zbirnik. **Solidity zbirnik** (angl. Solidity assembly oziroma Yul) je nizkonivojski programski jezik, v katerem programer neposredno komunicira s pomnilnikom, kar omogoča pisanje bolj optimiziranih in posledično cenejših aplikacij [13, 19].

Z vpeljavo pametnih pogodb smo v zadnjih letih priča vrsti varnostnih izzivov, saj so pametne pogodbe tarča napadalcev, ki iščejo načine za neavtorizirano spreminjanje njihovega stanja. Napadi na pametne pogodbe, napisane v Solidityju, postajajo vse pogostejši, kar ogroža varnost in stabilnost celotnega Ethereumovega ekosistema. Na pametnih pogodbah rešitev na Ethereumu z visokim prometom ali veliko kapitalizacijo je pogosto zaklenjenih več milijard dolarjev, zaradi česar so tovrstne pogodbe izjemno privlačne za napadalce [5].

V tem prispevku se osredotočamo na raziskovanje varnostnih izzivov, povezanih z EVM in Solidity pametnimi pogodbami. V drugem poglavju predstavimo EVM okolje. Nato se v tretjem poglavju osredotočimo na pregled napadov na platforme Ronin Network, Poly Network in Coincheck, ki so se zgodili v preteklosti, ter analiziramo njihove posledice in razpravljamo o možnih strategijah za preprečevanje takšnih incidentov v prihodnosti. V tretjem poglavju prikažemo primer napada na pametno pogodbo, napisano v zbirniku. Č lanek zaključimo s kratkim povzetkom vplivov varnostnih lukenj na tehnologijo veriženja blokov in povzamemo lasten pogled na prihajajoče izzive v prihodnosti.

2 OSNOVE ETHEREUMA IN EVM OKOLJA

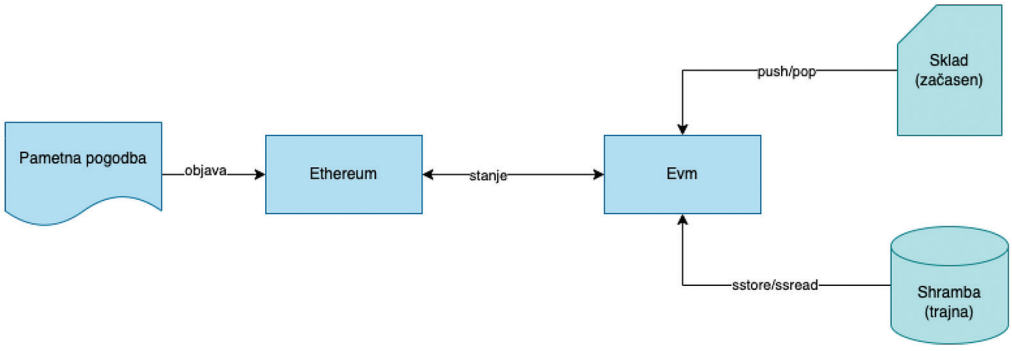
Takoj za Bitcoinom je Ethereum druga najbolj priljubljena kriptovaluta na svetu glede na tržno kapitalizacijo [7]. Ethereum omrežje je sestavljeno iz številnih **vozlišč** (angl. nodes), ki jih predstavljajo validatorji, povezani v omrežje. Vsak validator neodvisno izvaja operacije v EVM, kar omogoča delovanje pametnih pogodb. Ta decentralizirana narava omrežja zagotavlja njegovo varnost in odpornost proti napadom. Validatorji so odgovorni za preverjanje in potrjevanje transakcij, ki se zabeležijo v blokih na verigi [9].

Solidity programi se prevedejo v binarno kodo, ki se nato izvede znotraj EVM. Ta proces izvajanja vključuje tudi interakcijo z globalnim stanjem Ethereumovega omrežja, kar omogoča, da pametne pogodbe berejo in pišejo podatke, ki so shranjeni v vsakem validatorju znotraj Ethereumovega omrežja (septembra leta 2023 je imel Ethereum več kot 800 tisoč aktivnih validatorjev [14]).

EVM je osrednji del Ethereumovega ekosistema, ki omogoča izvajanje pametnih pogodb. Vsaka pametna pogodba je program, ki predstavlja deterministično dogovor med pogodbo in uporabniki. Ko je pogodba enkrat objavljena na verigi blokov, jo je nemogoče popravljati ali spreminjati. Interakcije s pogodbo uporabniki dosežejo s klicanjem funkcij pogodbe. Primer pametne pogodbe je program, ki ima funkcijo *izplacajNagrado*(*Dokaz p*), v katero uporabnik kot parameter poda dokaz, da je s svojim računalnikom pripomogel k treniranju nevronske mreže, pametna pogodba pa nato preveri verodostojnost podanega dokaza in izplača nagrado uporabniku.

Podatki v EVM se shranjujejo v dveh glavnih oblikah: v pomnilniku (angl. memory) in v shrambi (angl. storage), kot je prikazano na sliki 1. Pomnilnik je začasen in se izbriše med transakcijami, medtem ko je shramba trajna in se uporablja za shranjevanje stanja pametnih pogodb. Shema na sliki 1 prikazuje proces, ki se izvede na vsakem od validatorjev, rezultati tega procesa (spremenjeno stanje shrambe) pa se nato zapišejo v bloke. Vsaka operacija, ki se izvaja v EVM, stane določeno količino **goriva** (angl. gas). Gorivo je notranja enota za merjenje dela, ki se izvaja, in se uporablja za preprečevanje zlorab omrežja. Uporabniki plačajo transakcijske stroške v obliki goriva, ki se nato pretvori v Ethereumovo domačo valuto ETH. Nekatere transakcije, kot so na primer klici funkcij pametnih pogodb s kompleksno logiko, porabijo večje količine goriva in so zato dražje [21].

V zadnjem desetletju je zanimanje za decentralizirane aplikacije močno naraslo, pri čemer je Ethereum postal osrednja platforma za razvoj in izvajanje teh aplikacij. Ključni dejavnik v tej rasti so veliki protokoli in platforme, ki so bili razviti na Ethereumu in so postali ključni gradniki ekosistema. V tabeli 1 je predstavljenih sedem največjih rešitev glede na



Slika 1: **Shema izvajanja procesa znotraj EVM**

Tabela 1: **Tabela z osnovnimi finančnimi podatki o največjih rešitvah na Ethereumu [8]**

Rešitev	Uporaba	Vrednost žetona [$]	Zaklenjena vrednost [$]	Dnevni volumen [$]
LIDO	zastavljanje	2.43B	43.9B	17.32M
EigenLayer	zastavljanje	/	10.17B	/
Aave V3	izposoja	1.8B	7.76B	1.17M
MakerDAO	kovanec DAI	3.31B	6.67B	/
Uniswap V3	trgovanje	8.59B	5.07B	3.79B
Summer.fi	izposoja	5.26B	4.48B	951.95k
RocketPool V3	zastavljanje	582.7M	4.47B	404.39k

zaklenjeno vrednost (angl. total value locked – TVL) skupaj z vrednostjo žetona (angl. market capitalization – CAP) in dnevnim volumnom. Kot je razvidno iz tabele, je v nekaterih pametnih pogodbah zaklenjeno več milijard dolarjev. Kljub vsem varnostnim ukrepom, bi zlonamerni napadalec lahko odkril ranljivost, ki bi mu omogočila klicanje funkcij za prenašanje zaklenjenih sredstev. V tem primeru bi napadalec lahko ukradel ogromno vsoto denarja.

Izmed vseh pogodb na Ethereumu je 72,9% visoko aktivnih pogodb namenjenih funkcijam žetonov, kar predstavlja izjemno tržno kapitalizacijo v višini 12,7 milijarde dolarjev. Izmed teh je 60,4% preverjenih pogodb o žetonih, ki se pogosto uporabljajo za upravljanje delnic podjetja, na primer prek začetnih ponudb kovancev (angl. initial coin offering – ICO) [16]. Pogodbe tega tipa običajno uporabljajo preverjeni ERC-20 standard za implementacijo žetona, ki implementira osnovno funkcionalnost pošiljanja in prejemanja žetonov v 312 vrsticah kode Solidity, brez uporabe zbirnika. ERC-20 je najpopularnejša pametna pogodba na Etherumu – predstavlja kar 45.9% vseh objavljenih pametnih pogodb, vsak izmed prvih 92 ERC-20 žetonov ima skupno vrednost večjo od 1 milijarde dolarjev [20, 6]. ERC-20 standard, ki ga je razvilo in pregledalo eno izmed največjih podjetij za iskanje ranljivosti v pametnih pogodbah, OpenZeppelin, je v uporabi že od leta 2015 in ga večina razvijalcev obravnava kot popolnoma varno pametno pogodbo [17].

23% vseh Ethereumovih pogodb vsebuje vsaj 1 blok Solidity zbirnika [3]. Uporaba zbirnika je v primerjavi z uporabo navadne Solidity kode manj spominsko varna in zahtevnejša za razvijalce. Med razvijanjem pametnih pogodb je standardna praksa izogibanje neposrednemu dostopanju do pomnilnika, če to ni absolutno nujno. Ne glede na trud, vložen v preverjanje kode, napisane v zbirniku, izvajanje strojnih ukazov na pogodbi predstavlja varnostno luknjo.

Razvijalci vse pogosteje vključujejo zbirnik v svoje pogodbe, kar je povezano tudi z razvojem jezika Solidity, saj vsaka nova posodobitev v načinu dostopanja in pisanja v pomnilnik odpre možnosti za nove optimizacije. Uporaba zbirnika v pametnih pogodbah je začela strmo naraščati že leta 2015, ko je rast vrednosti žetona ETH povzročila, da so narasle tudi cene transakcij. Višja cena transakcij je sprožila

vse večjo potrebo po optimizaciji pametnih pogodb. Trend kaže, da razvijalci pri prevajanju pogodb, ki vsebujejo zbirnik, pogosto uporabljajo najnovejše različice prevajalnika Solidity. To kaže na potrebo po izkoriščanju najnovejših funkcionalnosti in izboljšav, ki jih prinašajo posodobitve jezika [3].

Pogodbe, ki vsebujejo zbirnik, imajo običajno več transakcij, edinstvenih klicateljev in vrstic kode v

primerjavi s pogodbami brez zbirnika. To nakazuje, da je zbirnik pogosteje uporabljen v pogodbah s kompleksno logiko, kjer je cena transakcij visoka. Toda prenosi žetonov so pogostejši v pogodbah, ki ne uporabljajo zbirnika. Z vidika potencialnega napadalca to pomeni, da bi ob morebitni varnostni luknji manj verjetno lahko klical funkcije, ki so odgovorne za prenos žetonov in s tem ukradel sredstva zaklenjena na pogodbi. Pomembno je omeniti, da so pogodbe z zbirnikom običajno večje od povprečnih pogodb, s povprečjem 248,94 vrstic kode v primerjavi s povprečno 62,36 vrsticami pri pogodbah brez zbirnika. Poleg tega so pogodbe, ki prenašajo nezamenljive žetone (angl. non-fungible tokens – NFT), nagnjene k uporabi zbirnika pogosteje kakor pogodbe, ki prenašajo ERC-20 žetone [3]. Zaradi večje varnosti razvijalci za svoj produkt pogosto objavijo dve ločeni pogodbi. Prva pogodba skrbi za interno logiko produkta, druga pa je ločen ERC-20 kovanec, ki nima nobene dodatne logike in ne uporablja zbirnika. Na tak način so stanja kovancev uporabnikov shranjena na ločeni pogodbi, do katere ne moremo neposredno dostopati iz okolja prve.

Več kot tri četrtine vseh blokov zbirnika (80,86%) vsebuje vsaj en ukaz za pridobivanje in shranje-

vanje informacij iz shrambe (SLOAD, SSTORE). V primeru napada ali napačnega delovanja zbirnika bi torej v večini primerov ogrozili tudi trajno stanje pogodbe, saj bi se spremenjene vrednosti v pomnilniku zapisale v shrambo. Operacije nadzora toka (ISZERO, JUMP) so prisotne le v 24,17% fragmentov, kar je pričakovano, saj preverjanje logičnih pogojev ni računsko zahtevno in ga ni smiselno prepisovati v zbirnik [3].

Citirana sorodna dela na področju analize varnosti Ethereumovih pametnih pogodb in Solidity zbirnika se osredotočajo na širšo analizo načina uporabe zbirnika. Z vidika varnosti je pomembno, da poznamo, kakšne so programerske prakse, kateri vzorci se pojavljajo in na kakšen način se uporabljao pametne pogodbe, ki vključujejo Solidity zbirnik. To delo se najprej osredotoča na opis preteklih napadov na decentralizirane protokole, v drugem delu pa opisujemo specifičen način napada na pogodbo v Solidity zbirniku z uporabo nedovoljene modifikacije vrednosti registra ter obrambo pred tovrstnim napadom.

3 PRETEKLI NAPADI NA DECENTRALIZIRANE PROTOKOLE

V tem poglavju obravnavamo tri odmevne primere napadov na decentralizirane protokole, ki so povzročili velike finančne izgube in opozorili na nekatere ključne ranljivosti v ekosistemu kriptovalut. Prvi primer je napad na Coincheck, japonsko borzo kriptovalut, ki je bila leta 2018 tarča napada zaradi pomanjkljivih varnostnih ukrepov in uporabe vročih denarnic. Drugi primer je napad na Poly Network leta 2021, kjer so hekerji izkoristili slabše upravljanje dostopnih pravic med pametnimi pogodbami. Tretji primer pa je napad na Ronin Network Bridge leta 2022, ko je prišlo do kompromitacije zasebnih ključev v modelu Proof of Authority. Vsi trije primeri poudarjajo ranljivosti decentraliziranih protokolov ter potrebo po strožjih varnostnih ukrepih pri shranjevanju zasebnih ključev in upravljanju dostopnih pravic.

3.1 Coincheck

Coincheck je vodilna borza s kriptovalutami na Japonskem. Napad, ki se je zgodil v začetku leta 2018, je povzročil, da je borza takrat utrpela za približno 533 milijonov dolarjev škode. Več kot 500 milijonov žetonov NEM je bilo prenesenih na 19 različnih naslovov napadalcev znotraj omrežja. Napadalci so uspeli napasti borzo preko elektronskih sporočil, ki so vsebovala viruse, kar jim je omogočilo dostop do zasebnih ključev borze. S pridobljenimi zasebnimi ključi je bil postopek kraje sredstev za napadalce izjemno enostaven, še posebej, ker Coincheck ni uporabljal avtorizacijskih pametnih pogodb ali tehnologije večih podpisov. Hkrati je podjetje vse kovance shranjevalo v isti vroči denarnici (tip denarnice, pri katerem je zasebni ključ shranjen neposredno v spominu naprave, ki izvede proces podpisovanja), kar je krajo sredstev še dodatno olajšalo. Coincheck je novembra 2018 po napadu spet začel ponujati polne storitve in še vedno posluje [4].

3.2 Poly Network Bridge

Podobno kot Ronin Network je tudi Poly Network decentraliziran most med različnimi omrežji. 10. av-

gusta 2021 je Poly Network poročal, da je neznani napadalec vdrl v pametno pogodbo omrežja, prenesel ekvi- valent približno 610 milijonov dolarjev (predvsem v žetonih ETH, BNB in USDC) ter jih premaknil na svoje denarnice. Po poročanju kibernetskega podjetja SlowMist in varnostnega raziskovalca Kelvina Fichterja je bil napad omogočen zaradi nepravilnega upravljanja dostopnih pravic med dvema pomembnima pame- tnima pogodbama znotraj Poly Networka. Prva je EthCrossChainManager, druga pa EthCrossChainData. EthCrossChainData je izjemno privilegirana pogodba, ki naj bi bila namenjena izključno klicem lastnikov. Razlog je, da je ta pogodba odgovorna za nastavitev in upravljanje seznama javnih ključev avtentikacijskih naslovov, ki upravljajo denarnice v osnovnih likvidnostnih bazenih. Ker je napadalcu uspelo poklicati funkcijo putCurEpochConPubKeyBytes znotraj EthCrossChainData, napad na privatni ključ avtentikacijskega naslova ni bil potreben. Napadalec je tako lahko preprosto zamenjal javni ključ avtentikacijskega naslova in s tem dobil pravice do razpolaganja s sredstvi v likvidnostnih bazenih [18, 12].

3.3 Ronin Network Bridge

Ronin Network je vmesno decentralizirano omrežje (most), zgrajeno na platformi EVM in uporabljeno v večjem omrežju Axie Infinity. 23. marca 2022 je v napadu na omrežje napadalec ukradel za približno 624 milijonov dolarjev sredstev v obliki žetonov ETH in USDC. Ključni dejavnik napada je bila kompromizacija zasebnega ključa, ki so jo napadalci izkoristili za ponarejanje lažnih dvigov in krajo sredstev iz omrežja. Napad je bil izveden preko modela Proof of Authority, ki je zahteval pet od devetih potrditev, ki jih je napadalec uspešno ponaredil in s tem pridobil nadzor nad omrežjem. Po napadu je ekipa omrežja napovedala ukrepe, kot je povečanje števila potrditev transakcij na osem od devetih validacij. Rešitev za zmanjšanje tveganja tovrstnega napada bi bila uporaba stroge varnostne prakse, kot je hranjenje ključev na denarnicah, ki niso povezane z internetom, in uporaba fizičnih večpodpisnih denarnic [18].

4 PRIKAZ NAPADA NA PAMETNO POGODBO NAPISANO V SOLIDITY ZBIRNIKU

V tem poglavju obravnavamo napad na pametno pogodbo UnsafeMemoryWrite, ki temelji na nepravilnem upravljanju spomina v zbirniku Solidity. Implementacija te pametne pogodbe ob pričakovani uporabi deluje brezhibno in pri prevajanju ne povzroča nobenih napak ali opozoril, vendar pa zlonamerna uporaba vstopnih parametrov omogoča nepooblaščeno spreminjanje shranjenih podatkov v pametni pogodbi. V praktičnem delu analiziramo, kako zloraba parametrov funkcije changeStoredValues omogoča napadalcu, da prepiše stanje uporabnika in si pridobi nedovoljeno finančno korist. Poleg tega predstavimo, kako se tovrstnim napadom lahko izognemo z ustreznimi obrambnimi tehnikami.

4.1 Metodologija

Za implementacijo pametne pogodbe UnsafeMemoryWrite je bil uporabljen Solidity zbirnik. Za razvoj in objavo pogodbe na testnem omrežju Sepolia je bilo upodabljano orodje Hardhat [11]. Postopek razvoja je vključeval uporabo Solidity zbirnika za simulacijo nedovoljenih operacij s spominom. Za simulacijo in analizo napada je bil uporabljen raziskovalec blokov Etherscan za spremljanje interakcij z verigo blokov ter orodje Hardhat Tracer za spremljanje stanja sklada in shrambe pogodbe med izvajanjem [10, 22]. Za analizo varnostnih lukenj pogodbe ter pridobitev strojne kode je bilo uporabljeno orodje Dedaub, napredno orodje za analizo ranljivosti pametnih pogodb [15]. Simulacija napada je vključevala pošiljanje zlonamernih parametrov funkciji changeStoredValues, kar je povzročilo nenamerno spremembo uporabnikovega stanja. Na podlagi teh opazovanj smo ugotovili kritične varnostne pomanjkljivosti, ki omogočajo napad. Na koncu smo s pomočjo analize delovanja pametne pogodbe prikazali načine za obrambo pred prikazanim napadom.

4.2 Pametna pogodba UnsafeMemoryWrite

Praktični del tega prispevka predstavlja implementacija pametne pogodbe UnsafeMemoryWrite, ki vsebuje Solidity zbirnik. Celotna koda pametne pogodbe je prikazana na sliki 2. Pametna pogodba z identično implementacijo je objavljena na testnem omrežju Sepolia [10]. Namen pogodbe je prikazati implementacijo zbirnika, ki v primeru pravilnih vhodnih podatkov deluje kot je pričakovano, vendar pa ob zlonamerno izbranih parametrih napadalcu omogoča nedovoljeno spreminjanje stanja. Ko pogodbo prevedemo in objavimo, nobeno od standardnih orodij ne javi opozorila ali napake. Implementacija je neodvisna od različice Solidity prevajalnika in deluje tudi v najnovejših različicah.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract UnsafeMemoryWrite {
    uint256 public userBalance = 1;

    function initializeValues(uint256 replacementStart, uint256 a, uint256 b) public {
        uint256 globalMemoryStart;
        assembly {
            globalMemoryStart := mload(0x40)
            mstore(add(mload(0x40), 0x00), add(sload(userBalance.slot), 1))

            mstore(add(mload(0x40), 0x20),
             ↪  0x583facc679f76c52a1127a11024efb42583facc679f76c52a1127a11024efb42)
            mstore(add(mload(0x40), 0x40),
             ↪  0x8d4e5f9b3a6c8e1f0b2d9c8a7e6f5d8c583facc679f76c52a1127a11024efb42)
            mstore(add(mload(0x40), 0x60),
             ↪  0x9a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d583facc679f76c52a1127a11024efb42)

            mstore(0x40, sub(add(mload(0x40), 0x80), replacementStart))
        }

        changeStoredValues(a, b);

        assembly {
            sstore(userBalance.slot, mload(globalMemoryStart))
        }
    }

    function changeStoredValues(uint256 a, uint256 b) public pure {
        assembly {
            mstore(add(mload(0x40), 0x00),
             ↪  0xe1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6583facc679f76c52a1127a11024efb42)
        }
        add(a, b);
    }

    function add(uint256 x, uint256 y) public pure returns (uint256) {
        uint256 memoryStart;
        uint256 res;
        assembly {
            memoryStart := mload(0x40)

            mstore(add(memoryStart, 0x00), add(x, y))
            res := mload(add(memoryStart, 0x00))

            memoryStart := sub(memoryStart, 0x20)
        }
        return res;
    }
}
```

Slika 2: **Izvorna koda UnsafeMemoryWrite**

4.3 Implementacija UnsafeMemoryWrite

Pogodba ima eno spremenljivko userBalance, ki predstavlja uporabnikovo stanje internega žetona. V resničnem primeru bi uporabnik na menjalnici lahko ta žeton zamenjal za drugo valuto, na primer ETH ali ameriške dolarje. Funkcija initializeValues najprej poveča uporabnikovo stanje za 1 (v praksi bi to lahko bila nagrada za računsko delo), potem pa v spomin zapiše trikrat po 32 bajtov spomina (na primer inicializacijski podatki za Pedersonovo zgoščevalno funkcijo). Na koncu funkcija spremeni vrednost registra 0x40 na vrednost parametra replacementStart tako, da se naslavljanje lokacije spremembe dogaja od konca proti začetku. Register 0x40 je odgovoren za shranjevanje kazalca na naslednji prosti bajt, kamor lahko alociramo spomin v skladu. Funkcija changeStoredValues zamenja blok bajtov na lokaciji replacementStart z drugimi vrednostmi in kliče funkcijo add. Funkcija add je standardna implementacija seštevanja v zbirniku.

4.4 Primer pravilnega delovanja

Interakcije s pogodbo so mogoče preko raziskovalcev blokov Etherscan ali Blockscout [10]. Č e na začetku kličemo funkcijo performCalculations s parametri replacementStart=0x20, a=10 in b=20, se v spomin zapišejo pravi podatki, zamenjava bloka bajtov se zgodi na predzadnjem mestu in rezultat funkcije add(10, 20) je 30. Na koncu izvajanja se je userBalance povečal z 1 na 2.

```
...
0x196: PUSH32    0x9a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d583facc679f76c52a1127a11024efb42
0x1b7: PUSH1     0x60
0x1b9: PUSH1     0x40
0x1bb: MLOAD
0x1bc: ADD
0x1bd: MSTORE
0x1be: DUP4
0x1bf: PUSH1     0x80
0x1c1: PUSH1     0x40
0x1c3: MLOAD
...
0x1c6: PUSH1     0x40
0x1c8: MSTORE
0x1c9: PUSH2     0x1d2
0x1cc: DUP4
0x1cd: DUP4
0x1ce: PUSH2     0xff
0x1d1: JUMP
0x1d2: JUMPDEST
0x1d3: DUP1
0x1d4: MLOAD
0x1d5: PUSH0
0x1d6: SSTORE
...
```

Slika 3: **Strojna koda MemoryWriteExample**

4.5 Napad na UnsafeMemoryWrite

Med izvajanjem pogodbe je na začetku v spominu shranjen userBalance (32B) in takoj za njim inicializacijske vrednosti (3x32B). Č e kot parameter replacementStart podamo vrednost 0x80, se bo register 0x40 nastavil tako, da bo ob klicu changeStoredValues kazalec na naslednji prosti bajt kazal na shranjeno vrednost uporabnikovega stanja. Ko se bo poklicala funkcija add, bo njen vmesni rezultat prepisal prejšnjo vrednost uporabnikovega stanja. Ker sta parametra a in b prosto nastavljiva, ju lahko napadalec nastavi na visoki vrednosti in na tak način nedovoljeno poveča svoje stanje. Klicanje funkcije initializeValues s parametri replacementStart=0x80, a=1000 in b=1000 spremeni stanje pogodbe tako, da je nova vrednost userBalance enaka 2000. Č e pogledamo strojno kodo prevedene pogodbe na sliki 3, vidimo, da se vrednost na 0x40 spremeni na vrednost podanega parametra ter da se ob koncu klicanja funkcij nova vrednost uporabnikovega stanja shrani v shrambo pogodbe z ukazom SSTORE. Prevedena strojna koda je bila iz omrežja Sepolia pridobljena s pomočjo orodja Dedaub – naprednega orodja za iskanje ranljivosti v pametnih pogodbah, ki prav tako ni javilo nobenega opozorila ali napake ob pregledu objavljene pametne pogodbe [15].

4.6 Obramba pred napadom

Obramba pred opisanim napadom je mogoča na več nivojih. Vsakič, ko je zaloga vrednosti parametra omejena, je potrebno z ukazom require preveriti, ali je parameter znotraj dovoljenih omejitev, preden ga začnemo uporabljati. Prav tako bi bila dobra praksa shranjevanje uporabnikovega stanja zunaj spomina, dostopnega na zbirniku, na primer v ločeni ERC-20 pogodbi, do katere znotraj okolja zbirnika nimamo dostopa. Zadnja rešitev varnostne luknje pa bi bila prepoved spreminjanja vrednosti registra 0x40 znotraj zbirnika ter izvajanje pisanja in branja iz spomina relativno na globalMemoryStart. Tak način programiranja zahteva več pozornosti razvijalcev, vendar pa eliminira možnost, da bi se v register 0x40 zapisala neveljavna vrednost.

5 SKLEPNE UGOTOVITVE IN ZAKLJUČEK

Raziskave varnostnih vprašanj, povezanih z EVM in Solidity pametnimi pogodbami, so ključne za izboljšanje varnosti in stabilnosti celotnega Ethereumovega ekosistema. Analiza preteklih napadov na decentralizirane platforme, kot so Coincheck, Poly Network Bridge in Ronin Network, ponuja pomembne uvide o možnih ranljivostih in potrebnih ukrepih za preprečevanje podobnih incidentov v prihodnosti

Napad na Coincheck je dokaz, da uspešen napad ni nujno posledica napake v zasnovi protokolov, ampak tudi socialnega inženiringa. Kljub varnostnim mehanizmom je šibkost vsake organizacije lahko tudi oseba, ki v trenutku nepozornosti odpre priponko sumljivega elektronskega sporočila. Napad velike razsežnosti, kakršen se je zgodil na Coincheck, je po navadi posledica kombinacije večih ranljivosti, ki jih napadalec izkoristi hkrati. Varnost je potrebno

izvajati na vseh nivojih, saj na tak način v primeru varnostne luknje na enem od nivojev morebitnega napadalca čim bolj omejimo.

V primeru Poly Network Bridge je bil ključni del napada nepreverjena povezava med posameznimi komponentami. Vsaka pametna pogodba, ki je sestavljala omrežje, je bila večkrat pregledana s strani različnih neodvisnih varnostnih podjetij. Med pregledi so bili spregledani načini interakcije med posameznimi pogodbami. Tudi če je sistem sestavljen iz komponent, ki so same po sebi varne, je potrebno vseeno preveriti tudi načine interakcije med njimi. Prav zaradi tega so vmesna omrežja, kot sta Poly Network Bridge in Ronin Network, tako pogosto tarča napadov. Med dvema sicer varnima omrežjema je pogosto, da je ravno vmesno omrežje najšibkejši člen.

Podobno kot napad na Coincheck, tudi napad na Ronin Network ni bil posledica slabo zasnovane arhitekture protokola, ampak načina upravljanja z njim. Dodatni varnostni ukrepi, kot je uporaba fizičnih večpodpisnih denarnic in zvišanje števila potrebnih potrditev, bi tudi v tem primeru lahko omejili škodo, ki jo je povzročil napad.

V pričujočem članku delu smo prikazali načine uporabe in izkoriščanja varnostnih lukenj v zbirniku

na primeru lastne pogodbe. V nasprotju z ostalimi opisanimi napadi je slednji posledica slabo zasnovane pametne pogodbe. Teoretični prikaz napada je razkril kompleksnost iskanja ranljivosti v zbirniku in razkril potencialno višjo stopnjo dovzetnosti za ranljivosti. Razumevanje kompleksnosti uporabe Solidity zbirnika in njegovih potencialnih tveganj je ključno za razvijalce in raziskovalcem ki se ukvarjajo z izboljšanjem varnosti decentraliziranih aplikacij na platformi Ethereum.

Naše nadaljnje delo zajema analizo uspešnosti odpravljanja tovrstnih napak z orodji za statično preverjanje kode, na primer Echinda. Poleg ranljivosti, povezanih z zbirnikom, je znotraj Ethereuma še vrsta drugih varnostnih vprašanj, ki jih v tem prispevku nismo naslovili, na primer izkoriščanje urejenosti transakcij v blokih, manipulacija časa bloka, ponovni vstop med klicanjem transakcije in napadi na orakle. Skupaj s stalnim razvojem in izboljšavami varnostnih praks so varnostni pregledi in teoretični prikazi napadov temeljno orodje za odkrivanje ranljivosti in reševanje varnostnih izzivov.

V prihodnosti bo tehnologija veriženja blokov postala še pomembnejša, saj se bo njena uporaba širila v različne industrije, vključno s finančnimi storitvami, dobavnimi verigami, zdravstvom in upravljanjem identitete. S tem povečanjem uporabe se bo neizogibno povečala tudi vrednost sredstev, ki jih upravljajo decentralizirani protokoli, kar bo hkrati pritegnilo več pozornosti napadalcev. Pričakovati je torej, da se bodo napadi na infrastrukturo verig blokov množili, postajali bodo bolj sofisticirani in ciljali bodo na bolj kompleksne ranljivosti. To poudarja nujnost nadaljnjega razvoja varnostnih praks in orodij za preprečevanje napadov, kot tudi rednega izvajanja varnostnih pregledov in revizij kode. Le s proaktivnim pristopom k varnosti lahko ohranimo zaupanje v varnost protokolov in zagotovimo stabilnost ter integriteto celotnega Ethereum ekosistema.

LITERATURA

[1] Vitalik Buterin. Ethereum: Platform review opportunities and challenges for private and consortium blockchains. *Opportunities and challenges for private and consortium blockchains*, 2016.

[2] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013.

[3] Stefanos Chaliasos, Arthur Gervais, and Benjamin Livshits. A study of inline assembly in solidity smart contracts. *Proc. ACM Program. Lang.*, 6(OOPSLA2), October 2022.

[4] Ben Charoenwong and Mario Bernardi. A decade of cryptocurrency 'hacks': 2011 – 2021. October 1 2021. Available at SSRN: https://ssrn.com/abstract=3944435 or http://dx.doi.org/10.2139/ ssrn.3944435.

[5] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Comput. Surv.*, 53(3), jun 2020.

[6] Coingecko. https://www.coingecko.com/. Dostopano: 2. Maj 2024.

[7] CoinMarketCap. Coinmarketcap. https://coinmarketcap.com/. Dostopano: 2. Maj 2024.

[8] Dapp radar. https://dappradar.com/rankings/defi/chain/ethereum. Dostopano: Apr 29, 2024.

[9] Ethereum. https://ethereum.org/en/. Dostopano: 2. Maj 2024.

[10] Etherscan. https://sepolia.etherscan.io/address/0xf72a4939282D4E947dA521bAA340186d8966DeEc, če navedeni naslov ne deluje uporabiti https://eth-sepolia.blockscout.com/address/ 0xf72a4939282D4E947dA521bAA340186d8966DeEc.

[11] Nomic Foundation. Hardhat: Ethereum development environment for professionals, 2024.

[12] Tommaso Gagliardoni. The poly network hack explained, August 12 2021. Research at Kudelski Security.

[13] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, and Grigore Rosu. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 204–217, 2018.

[14] Christine Kim. The most pressing issue on ethereum is validator size growth. https://www.coindesk.com/consensus-magazine/2023/09/29/ the-most-pressing-issue-on-ethe-

reum-is-validator-size-growth/, 2023. Dostopano: Sep 29, 2023.
[15] Oliver Lutz, Huili Chen, Hossein Fereidooni, Christoph Sendner, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. ESCORT: ethereum smart contracts vulnerability detection using deep neural network and transfer learning. *CoRR*, abs/2103.12607, 2021.
[16] Gustavo A Oliva, Ahmed E Hassan, and Zheng Ming Jiang. An exploratory study of smart contracts in the ethereum blockchain platform. *Empirical Software Engineering,* 25:1864–1904, 2020.
[17] OpenZeppelin. OpenZeppelin Contracts – ERC20.sol. https://github.com/OpenZeppelin/ openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol, Dostopano: Apr 29, 2024.
[18] Jason Scharfman. *Decentralized Finance (DeFi) Fraud and Hacks: Part 2*, pages 97–110. Springer International Publishing, Cham, 2023.
[19] Solidity Assembly Dokumentacija. https://docs.soliditylang.org/en/latest/assembly.html. Do- stopano: 2. Maj 2024.
[20] Friedhelm Victor and Bianca Katharina Lüders. Measuring ethereum-based erc20 token networks. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers*, page 113–129, Berlin, Heidelberg, 2019. Springer-Verlag.
[21] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić. Blockchain technology, bitcoin, and ethereum: A brief overview. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6, 2018.
[22] Zemse. Hardhat tracer. https://github.com/zemse/hardhat-tracer, 2023. Dostopano: 2024-09-18.

▪

Gal Gantar je absolvent dodiplomskega interdisciplinarnega študija računalništva in informatike. Njegovo raziskovalno delo se osredotoča na kriptografijo, predvsem na dokaze brez razkritja znanja. Prav tako raziskuje področja umetne inteligence, s posebnim poudarkom na globokem učenju na grafih. Strokovno je aktiven na področju razvoja decentraliziranih protokolov in pametnih pogodb.

▪

Matevž Pesek je docent in raziskovalec na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer je diplomiral (2012) in doktoriral (2018). Od leta 2009 je član Laboratorija za računalniško grafiko in multimedije. Od leta 2024 izvaja predmet Varnost programov.