// HALBORN

Taraxa - Node and EVM

L1 Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 1st, 2022 - August 31st, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/22/2022	Hossam Mohamed
0.2	Document Additions	07/13/2022	Erlantz Saenz
0.3	Draft Review	07/31/2022	Gabi Urrutia
0.4	Document Additions	08/19/2022	Chris Meistre
0.5	Draft Review	09/01/2022	Erlantz Saenz
0.6	Draft Review	09/02/2022	Gabi Urrutia
1.0	Remediation Plan	11/07/2022	Hossam Mohamed
1.1	Remediation Plan Edit	11/07/2022	Erlantz Saenz
1.2	Remediation Plan Edit	11/08/2022	Chris Meistre
1.3	Remediation Plan Edit	11/23/2022	Erlantz Saenz
1.4	Remediation Plan Review	11/29/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Erlantz Saenz	Halborn	erlantz.saenz@halborn.com
Hossam Mohamed	Halborn	hossam.mohamed@halborn.com
Chris Meistre	Halborn	Chris.Meistre@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Taraxa engaged Halborn to conduct a security assessment on taraxa-node and Taraxa-EVM codebase from June 1st, 2022 to August 31st.

1.2 AUDIT SUMMARY

The team at Halborn was provided a timeline for the engagement and assigned two full-time security engineers to audit the security of the assets in scope. The engineers are blockchain and smart contract security experts with advanced penetration testing, smart contract hacking, and in-depth knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Identify potential security issues within taraxa-node and Taraxa-EVM.

In summary, Halborn identified multiple security risks that were mostly addressed by the Taraxa team.

1.3 TEST APPROACH & METHODOLOGY

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk

level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The assessment was scoped for the following projects details:

- Taraxa-node (commit ID: d091acbb20853c629a8b04ea11dcf32241e1c897)
- Taraxa-EVM (commit ID: a23dbf14f9cf8513f1bde13757e2d9b27cf2db8c)

The main particular components and libraries under review were:
- RPC
- P2P
- Cryptography Signatures
- Keys Management
- Accounts and transactions
- Consensus
- Storage

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	0	5	2	7

LIKELIHOOD

IMPACT

				(HAL-01) (HAL-02)
		(HAL-04) (HAL-05) (HAL-06) (HAL-07) (HAL-08)		
	(HAL-09) (HAL-10)			
(HAL-03) (HAL-11) (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
NODE - VOTES - DENIAL OF SERVICE	Critical	SOLVED - 08/16/2022
NODE - RPC - NO AUTHENTICATION REQUIRED	Critical	SOLVED - 08/31/2022
NODE - BLOCK QUEUE WARNING WILL BLOCK INSTEAD OF WARNING	Informational	SOLVED - 07/28/2022

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
EVM - UNFILTERED PARAMETER ALLOWED TO EXECUTE COMMANDS ON THE HOST	Medium	SOLVED - 07/20/2022
EVM - LACK OF RETURN ERROR	Medium	SOLVED - 07/20/2022
EVM - LACK OF SIZE CHECK - OUT-OF-BOUNDS	Medium	SOLVED - 07/20/2022
EVM - INCORRECT ‘NIL‘ VALUE RETURNED ON AN ERROR	Medium	SOLVED - 11/07/2022
EVM - MULTIPLE OUTDATED MODULES	Medium	SOLVED - 11/07/2022
EVM - ERROR VALUE EVALUATED BUT NOT APPLIED	Low	SOLVED - 11/07/2022
EVM - NO ‘ERR‘ VARIABLE EVALUATION PRIOR TO AN OPERATION	Low	SOLVED - 11/07/2022
EVM - IMPLICIT MEMORY ALIASING IN LOOP	Informational	PARTIALLY SOLVED
EVM - LACK OF DEFAULT CLAUSE ON SWITCH STATEMENT	Informational	SOLVED - 11/23/2022
EVM - INSECURE RANDOM NUMBER GENERATOR	Informational	SOLVED - 11/07/2022
EVM - COMPARE INSTEAD OF EQUAL	Informational	SOLVED - 11/07/2022
EVM - MULTIPLE TO-DO COMMENTS FOUND ON THE CODE	Informational	ACKNOWLEDGED
EVM - PANIC IS USED FOR ERROR HANDLING	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

TARAXA NODE

4.1 (HAL-01) VOTES - DENIAL OF SERVICE - CRITICAL

Description:

Uncontrolled resource consumption and out-of-memory (OOM) vulnerability was observed within taraxa-node code. This could be exploited in a Denial of Service/Distributed Denial of Service (DoS/DDoS) attack against the taraxa node via p2p messaging.

The attack could be performed by a peer rapidly sending multiple unverifiable votes, which would be handled later on via VotePacketHandler. Calling the addUnverifiedVote function would append the votes to an unverified_votes_ memory map that it would continue to grow without limit, causing the node to crash.

Code Location:

The main cause of the problem was within the VotePacketHandler packet, and unverified votes were stored in the classic memory map. The attacker could send a vote packet containing 1000 votes.

Listing 1: /libraries/core_libs/network/src/tarcap/packets_handlers/vote_packet_handler.cpp (Line 57)

```
23
24 void VotePacketHandler::process(const PacketData &packet_data,
↳ const std::shared_ptr<TaraxaPeer> &peer) {
25   std::vector<std::shared_ptr<Vote>> votes;
26   const auto count = packet_data.rlp_.itemCount();
27   for (size_t i = 0; i < count; i++) {
28     auto vote = std::make_shared<Vote>(packet_data.rlp_[i].data().
↳ toBytes());
29     const auto vote_hash = vote->getHash();
30     LOG(log_dg_) << "Received PBFT vote " << vote_hash;
31
32     const auto vote_round = vote->getRound();
33     const auto current_pbft_round = pbft_mgr_->getPbftRound();
34
```

```
35      // Check reward vote
36      if (vote_round < current_pbft_round) {
37        // Synchronization point in case multiple threads are
↳ processing the same vote at the same time
38        if (!seen_votes_.insert(vote_hash)) {
39          LOG(log_dg_) << "Received vote " << vote_hash << " (from "
↳  << packet_data.from_node_id_.abridged()
40                       << ") already seen.";
41        } else if (vote_mgr_->addRewardVote(vote)) {
42          // As peers have small caches of known votes. Only mark
↳ gossiping votes
43          peer->markVoteAsKnown(vote_hash);
44          votes.push_back(std::move(vote));
45        }
46        continue;
47      }
48
49      // Votes vote_round >= current_pbft_round
50      // Synchronization point in case multiple threads are
↳ processing the same vote at the same time
51      if (!seen_votes_.insert(vote_hash)) {
52        LOG(log_dg_) << "Received PBFT vote " << vote_hash << " (
↳ from " << packet_data.from_node_id_.abridged()
53                     << ") already seen.";
54        continue;
55      }
56
57      // Adds unverified vote into queue
58      if (vote_mgr_->voteInVerifiedMap(vote) || !vote_mgr_->
↳ addUnverifiedVote(vote)) {
59        LOG(log_dg_) << "Received PBFT vote " << vote_hash << " (
↳ from " << packet_data.from_node_id_.abridged()
60                     << ") already saved in (un)verified queues.";
61        continue;
62      }
63      // Do not mark it before, as peers have small caches of known
↳ votes. Only mark gossiping votes
64      peer->markVoteAsKnown(vote_hash);
65      votes.push_back(std::move(vote));
66    }
67
68    onNewPbftVotes(std::move(votes));
69  }
```

Listing 2: /libraries/core_libs/network/src/tarcap/packets_handlers/vote_packet_handler.cpp (Lines 72,66)

```
58
59 bool VoteManager::addUnverifiedVote(std::shared_ptr<Vote> const&
↳ vote) {
60   uint64_t pbft_round = vote->getRound();
61   const auto& hash = vote->getHash();
62   vote->getVoterAddr();  // this will cache object variables -
↳ speed up
63
64   {
65     UniqueLock lock(unverified_votes_access_);
66     if (auto found_round = unverified_votes_.find(pbft_round);
↳ found_round != unverified_votes_.end()) {
67       if (!found_round->second.insert({hash, vote}).second) {
68         LOG(log_dg_) << "Vote " << hash << " is in unverified map
↳ already";
69         return false;
70       }
71     } else {
72       std::unordered_map<vote_hash_t, std::shared_ptr<Vote>> votes
↳ {std::make_pair(hash, vote)};
73       unverified_votes_[pbft_round] = std::move(votes);
74     }
75   }
76   LOG(log_nf_) << "Add unverified vote " << vote->getHash().
↳ abridged();
77
78   return true;
79 }
80
```

Listing 3: /libraries/core_libs/consensus/include/vote_manager/vote_manager.hpp (Line 336)

```
334
335   // <pbft round, <vote hash, vote>>
336   std::map<uint64_t, std::unordered_map<vote_hash_t, std::
↳ shared_ptr<Vote>>> unverified_votes_;
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Use ExpirationCache to store unverified votes instead of the classic C++ map.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by removing unverified_votes_ and refactoring the vote_manager class. ece157723262507f0209942fc573b80f5e436eb6

4.2 (HAL-02) RPC - NO AUTHENTICATION REQUIRED - CRITICAL

Description:

The RPC Server did not implement authentication or level-based access control, which allowed anyone with network level access to create raw transactions using the send_coin_transaction call.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Implement RPC Authentication and whitelisting for users, where users must have privileges to call specifics group of RPC calls.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by removing the eth_sendTransaction call and disabling the RPC by default.
1914

4.3 (HAL-03) BLOCK QUEUE WARNING WILL BLOCK INSTEAD OF WARNING - INFORMATIONAL

Description:

There was a typo in the name within the configuration file. The user could supply the max_block_queue_warn number, expecting to generate warning logs. Instead, it would block the addition of new blocks to the queue.

Listing 4: libraries/core_libs/node/src/node.cpp (Line 109)

```
17 dag_blk_mgr_ =
18       std::make_shared<DagBlockManager>(node_addr, conf_.chain.
 ↳ sortition, conf_.chain.dag, db_, trx_mgr_, final_chain_,
19                                         pbft_chain_, conf_.
 ↳ max_block_queue_warn, conf_.max_levels_per_period);
20   dag_mgr_ = std::make_shared<DagManager>(dag_genesis_block_hash,
 ↳ node_addr, trx_mgr_, pbft_chain_, dag_blk_mgr_, db_,
21                                           conf_.is_light_node,
 ↳ conf_.light_node_history, conf_.max_levels_per_period,
22                                           conf_.dag_expiry_limit);
23   vote_mgr_ = std::make_shared<VoteManager>(node_addr, db_,
 ↳ final_chain_, next_votes_mgr_);```
24
```

Listing 5: libraries/core_libs/consensus/src/dag/dag_block_manager.cpp (Lines 113,114)

```
 99 DagBlockManager::InsertAndVerifyBlockReturnType DagBlockManager::
 ↳ insertAndVerifyBlock(DagBlock &&blk) {
100   if (isDagBlockKnown(blk.getHash())) {
101     LOG(log_dg_) << "Trying to push new unverified block " << blk.
 ↳ getHash().abridged()
102                  << " that is already known, skip it";
103     return InsertAndVerifyBlockReturnType::AlreadyKnown;
104   }
105
106   // Mark block as seen - synchronization point in case multiple
 ↳ threads are processing the same block at the same time
```

```
107   if (!markDagBlockAsSeen(blk)) {
108     LOG(log_dg_) << "Trying to push new unverified block " << blk.
 ↳ getHash().abridged()
109                  << " that is already marked as known, skip it";
110     return InsertAndVerifyBlockReturnType::AlreadyKnown;
111   }
112
113   if (queue_limit_ > 0) {
114     if (const auto queue_size = getDagBlockQueueSize();
 ↳ queue_limit_ < queue_size) {
115       LOG(log_er_) << "Block queue large. Verified queue: " <<
 ↳ queue_size << "; Limit: " << queue_limit_;
116     }
117     return InsertAndVerifyBlockReturnType::BlockQueueOverflow;
118   }
119   const auto verified = verifyBlock(blk);
120   if (verified == InsertAndVerifyBlockReturnType::
 ↳ InsertedAndVerified) {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Rename max_block_queue_warn to max_block_queue.

Remediation Plan:

SOLVED: The Taraxa team solved the issue within the new dag block refactoring pull request.
1884

TARAXA EVM

5.1 (HAL-04) UNFILTERED PARAMETER ALLOWED TO EXECUTE COMMANDS ON THE HOST - MEDIUM

Description:

The SolidityVersion function located in common/compiler/solidity.go was detected to be vulnerable to uncontrolled command execution. The input parameter used by the function was not correctly sanitized, allowing an attacker to abuse the functionality and execute commands in the underlying Operating System.

Code Location:

- Taraxa-evm/common/compiler/solidity.go

Listing 6: Taraxa-evm/common/compiler/solidity.go (Lines 89-92,94,96)

```
88 // SolidityVersion runs solc and parses its version output.
89 func SolidityVersion(solc string) (*Solidity, error) {
90     if solc == "" {
91         solc = "solc"
92     }
93     var out bytes.Buffer
94     cmd := exec.Command(solc, "--version")
95     cmd.Stdout = &out
96     err := cmd.Run()
97     if err != nil {
98         return nil, err
99     }
100    matches := versionRegexp.FindStringSubmatch(out.String())
101    if len(matches) != 4 {
102        return nil, fmt.Errorf("can't parse solc version %q", out.
 ↳ String())
103    }
104    s := &Solidity{Path: cmd.Path, FullVersion: out.String(),
 ↳ Version: matches[0]}
105    if s.Major, err = strconv.Atoi(matches[1]); err != nil {
106        return nil, err
```

```
107     }
108     if s.Minor, err = strconv.Atoi(matches[2]); err != nil {
109         return nil, err
110     }
111     if s.Patch, err = strconv.Atoi(matches[3]); err != nil {
112         return nil, err
113     }
114     return s, nil
115 }
```

Proof of Concept:

Listing 7: PoC.go (Line 11)

```
 1
 2 package main
 3
 4 import (
 5         "os"
 6         "github.com/Taraxa-project/taraxa-evm/common/compiler"
 7 )
 8
 9 func main() {
10         //Call to the SolidityVersion function
11         compiler.SolidityVersion(os.Args[1])
12 }
```

```
root@de6cde168fcc:~/.go/src/github.com/Taraxa-project
/taraxa-evm# go run poc.go "curl"
curl 7.81.0 (x86_64-pc-linux-gnu) libcurl/7.81.0 OpenSSL/3.0.2 zl
ib/1.2.11 brotli/1.0.9 zstd/1.4.8 libidn2/2.3.2 libpsl/0.21.0 (+l
ibidn2/2.3.2) libssh/0.9.6/openssl/zlib nghttp2/1.43.0 librtmp/2.
3 OpenLDAP/2.5.12
Release-Date: 2022-01-05
Protocols: dict file ftp ftps gopher gophers http https imap imap
s ldap ldaps mqtt pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smt
ps telnet tftp
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTPS-proxy
 IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS
-SRP UnixSockets zstd
```

Figure 1: Command execution invoking curl command as an example.

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Avoid using the os/exec library whenever possible. This would allow an attacker to interact and execute commands on the underlying Operating System when it is not used correctly. However, filtering the input parameters would make the implementation of the function more secure.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by removing the solidity.go file. 5abe0c4dbcc94d5ee1b991fa9efcf15af16502a1

5.2 (HAL-05) LACK OF RETURN ERROR - MEDIUM

Description:

No return statements were found in the function that would return an error, due to the lack of a default statement in the Switch clause. In case another method makes use of this function, the error would never be caught, triggering undefined behavior in the EVM and consequently in the Node.

Code Location:

- Taraxa-evm/accounts/abi/abi.go

Listing 8: Taraxa-evm/accounts/abi/abi.go (Lines 110,93,132)

```
92 // UnmarshalJSON implements json.Unmarshaler interface
93 func (abi *ABI) UnmarshalJSON(data []byte) error {
94     var fields []struct {
95         Type      string
96         Name      string
97         Constant  bool
98         Anonymous bool
99         Inputs    []Argument
100        Outputs   []Argument
101    }
102
103    if err := json.Unmarshal(data, &fields); err != nil {
104        return err
105    }
106
107    abi.Methods = make(map[string]Method)
108    abi.Events = make(map[string]Event)
109    for _, field := range fields {
110        switch field.Type {
111        case "constructor":
112            abi.Constructor = Method{
113                Inputs: field.Inputs,
114            }
```

```
115         // empty defaults to function according to the abi spec
116         case "function", "":
117             abi.Methods[field.Name] = Method{
118                 Name:    field.Name,
119                 Const:   field.Constant,
120                 Inputs:  field.Inputs,
121                 Outputs: field.Outputs,
122             }
123         case "event":
124             abi.Events[field.Name] = Event{
125                 Name:      field.Name,
126                 Anonymous: field.Anonymous,
127                 Inputs:    field.Inputs,
128             }
129         }
130     }
131
132     return nil
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Include a Default statement that could be triggered on error, and update the return value accordingly.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by adding a default clause with an error message, which would be triggered in case the switch statement could not satisfy any of the offered options. 95f111ec8f9812f2174db0abd97af98c95e5ea77

5.3 (HAL-06) LACK OF SIZE CHECK - OUT-OF-BOUNDS - MEDIUM

Description:

The affected functions did not check the size of the array before performing the access. The function could trigger an Out-of-Bounds state and consequent Panic when trying to access a non-existent position.

Code Location:

- Taraxa-evm/taraxa/util/bin/index.go:ENC_b_endian_compact_64
- Taraxa-evm/taraxa/util/bin/index.go:DEC_b_endian_compact_64
- Taraxa-evm/taraxa/util/bin/index.go:ENC_b_endian_64
- Taraxa-evm/taraxa/util/bin/index.go:DEC_b_endian_64

Listing 9: Taraxa-evm/taraxa/util/bin/index.go (Lines 65-66)

```
64 func DEC_b_endian_64(b []byte) uint64 {
65     return uint64(b[0])<<56 | uint64(b[1])<<48 | uint64(b[2])<<40
↳ | uint64(b[3])<<32 |
66         uint64(b[4])<<24 | uint64(b[5])<<16 | uint64(b[6])<<8 |
↳ uint64(b[7])
67 }
```

Proof of Concept:

```
root@7caa7c43d3a4:~/test/mehmeh/crashers.bk# cat 39dfa55283318d31afe5a3ff4a0e3253e2045e43.quoted
"0000"
root@7caa7c43d3a4:~/test/mehmeh/crashers.bk# cat 39dfa55283318d31afe5a3ff4a0e3253e2045e43.output
panic: runtime error: index out of range [4] with length 4

goroutine 1 [running]:
github.com/Taraxa-project/taraxa-evm/taraxa/util/bin.DEC_b_endian_64(...)
        /root/test/src/github.com/Taraxa-project/taraxa-evm/taraxa/util/bin/index.go:66
_/root/test/mehmeh.Fuzz7(0x4002ac4000, 0x4, 0x4, 0x4)
        /root/test/mehmeh/poc.go:30 +0xb2
go-fuzz-dep.Main(0xc00008ef48, 0x1, 0x1)
        go-fuzz-dep/main.go:36 +0x1ad
main.main()
        _/root/test/mehmeh/go.fuzz.main/main.go:15 +0x52
```

Figure 2: Out-of-Bound error triggered by go-fuzz fuzzer.

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Check the size of the array before performing bitwise operations. As a general rule, the size of the array should be checked before any operation, so as not to trigger unhandled exceptions.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by adding a default clause with an error message, which would be triggered in case the switch statement could not satisfy any of the offered options. 85b89e5aed7f31f64cabe6e0dbd1bc8f1a1fc904

5.4 (HAL-07) INCORRECT ‘NIL‘ VALUE RETURNED ON AN ERROR - MEDIUM

Description:

The exposed function did not return a correct error value when an error was triggered within the function. This could lead to unexpected behaviors in other functions that make use of the errors returned by `Run()`. Returning a `nil` value in the error variable would not propagate the error.

Code Location:

Listing 10: Taraxa-evm/core/vm/contracts.go (Lines 115,117-118,122)

```
 99 func (c ecrecover) Run(ctx CallFrame, evm *EVM) ([]byte, error) {
100     const ecRecoverInputLength = 128
101     input := ctx.Input
102     input = common.RightPadBytes(input, ecRecoverInputLength)
103     // "input" is (hash, v, r, s), each 32 bytes
104     // but for ecrecover we want (r, s, v)
105
106     r := new(big.Int).SetBytes(input[64:96])
107     s := new(big.Int).SetBytes(input[96:128])
108     v := input[63] - 27
109
110     // tighter sig s values input homestead only apply to tx sigs
111     if !allZero(input[32:63]) || !crypto.ValidateSignatureValues(v
 ↳ , r, s, false) {
112         return nil, nil
113     }
114     // v needs to be at the end for libsecp256k1
115     pubKey, err := crypto.Ecrecover(input[:32], append(input
 ↳ [64:128], v))
116     // make sure the public key is a valid one
117     if err != nil {
118         return nil, nil
119     }
120
121     // the first byte of pubkey is bitcoin heritage
```

```
122     return common.LeftPadBytes(keccak256.Hash(pubKey[1:])[12:],
 ↳ 32), nil
123 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

When an error value is returned within the function, it must be sent to the rest of the affected functions to handle this unexpected behavior.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by adding an error message to the return value. 41cd2d7fdf855627f1b08209b6c804bf1bdd3f54

5.5 (HAL-08) MULTIPLE OUTDATED MODULES - MEDIUM

Description:

As part of the security audit process, an automated verification of project dependencies was performed. As a result, multiple vulnerable or outdated modules were found, but the affected dependencies were not exploited by auditors. However, this could be exploited if an attacker gains enough privileges or reach certain conditions.

Results:

Module	Version	Patched version
/aead/siphash	v1.0.1	
/btcsuite/btcd	v0.20.1-beta	v0.23.1
/btcsuite/btclog	v0.0.0-20170628155309	
/btcsuite/btcutil	v0.0.0-20190425235716	v1.0.2
/btcsuite/go-socks	v0.0.0-20170105172521	
/btcsuite/goleveldb	v0.0.0-20160330041536	v1.0.0
/btcsuite/snappy-go	v0.0.0-20151229074030	v1.0.0
/btcsuite/websocket	v0.0.0-20150119174127	
/btcsuite/winsvc	v1.0.0	
/davecgh/go-spew	v1.1.1	
/emicklei/dot	v0.10.2	v1.0.0
/fjl/gencodec	v0.0.0-20191126094850	v0.0.0-20220412091415
/fsnotify/fsnotify	v1.4.7	v1.5.4
/garslo/gogen	v0.0.0-20170306192744	v0.0.0-20170307003452
/golang/protobuf	v1.2.0	v1.5.2
/hpcloud/tail	v1.0.0	
/jessevdk/go-flags	v0.0.0-20141203071132	v1.5.0
/jrick/logrotate	v1.0.0	
/k0kubun/go-ansi	v0.0.0-20180517002512	
/kkdai/bstream	v0.0.0-20161212061736	v1.0.0

Module	Version	Patched version
/kr/pty	v1.1.1	v1.1.8
/kr/text	v0.1.0	v0.2.0
/kylelemons/godebug	v0.0.0-20170224010052	v1.1.0
/mattn/go-isatty	v0.0.12	v0.0.14
/mattn/go-runewidth	v0.0.9	v0.0.13
/mitchellh/colorstring	v0.0.0-20190213212951	
/niemeyer/pretty	v0.0.0-20200227124842	
/onsi/ginkgo	v1.10.3	v1.16.5
/onsi/gomega	v1.7.1	v1.20.0
/otiai10/copy	v1.2.0	v1.7.0
/otiai10/curr	v1.0.0	
/otiai10/mint	v1.3.1	v1.3.3
/pmezard/go-difflib	v1.0.0	
/schollz/progressbar/v3	v3.3.3	v3.9.0
/stretchr/objx	v0.1.0	v0.4.0
/stretchr/testify	v1.6.1	v1.8.0
/tecbot/gorocksdb	v0.0.0-20191217155057	
/x/crypto	v0.0.0-20200221231518	v0.0.0-20220722155217
/x/net	v0.0.0-20190620200207	v0.0.0-20220728211354
/x/sync	v0.0.0-20190423024810	v0.0.0-20220722155255
/x/sys	v0.0.0-20200223170610	v0.0.0-20220731174439
/x/text	v0.3.0	v0.3.7
/x/tools	v0.0.0-20191126055441	v0.1.12
/x/xerrors	v0.0.0-20190717185122	v0.0.0-20220609144429
/check.v1	v1.0.0-20200227125254	v1.0.0-20201130134442
/fsnotify.v1	v1.4.7	
/tomb.v1	v1.0.0-20141024135613	
/yaml.v2	v2.2.8	v2.4.0
/yaml.v3	v3.0.0-20200313102051	v3.0.1

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Patch or update exposed modules where possible, or reduce the attack surface so that it is impossible for an attacker to exploit this security flaw.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by applying the proposed recommendation. No vulnerable packages were found on the specified date. bee07cd1316adabb406ab39edfc2cee7d524a840

5.6 (HAL-09) ERROR VALUE EVALUATED BUT NOT APPLIED - LOW

Description:

The affected functions contained error checking in them. However, the errors were not handled properly and allowed the functions to continue to run. Even when the error clauses were modified accordingly, execution was not stopped by return statements.

When execution continues after an error, it could take advantage of unexpected behavior in linked functions that could not handle returned errors correctly.

Code Location:

Listing 11: Taraxa-evm/Common/hexutil/hexutil.go (Lines 67-68,71)

```
59 // Decode decodes a hex string with 0x prefix.
60 func Decode(input string) ([]byte, error) {
61     if len(input) == 0 {
62         return nil, ErrEmptyString
63     }
64     if !has0xPrefix(input) {
65         return nil, ErrMissingPrefix
66     }
67     b, err := hex.DecodeString(input[2:])
68     if err != nil {
69         err = mapError(err)
70     }
71     return b, err
72 }
```

Listing 12: Taraxa-evm/Common/hexutil/hexutil.go (Lines 97-98,101)

```
92 unc DecodeUint64(input string) (uint64, error) {
93     raw, err := checkNumber(input)
94     if err != nil {
95         return 0, err
```

```
96      }
97      dec, err := strconv.ParseUint(raw, 16, 64)
98      if err != nil {
99          err = mapError(err)
100     }
101     return dec, err
102 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Evaluate the error clause and apply the return values accordingly. In case an error is discovered, the return value should be adapted, it is recommended not to return the results of the operation.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by adding a return statement in case an error was detected. b4cedfde975bcd28d56203ecd8d59f75becb5936

5.7 (HAL-10) NO ‘ERR‘ VARIABLE EVALUATION PRIOR TO AN OPERATION - LOW

Description:

The err value of a function was not evaluated before an operation. If this error value is not evaluated properly, the value returned from the function could lead to multiple security issues, such as Out-of-Bounds and several unexpected states.

Code Location:

- Taraxa-evm/core/vm/evm.go

Listing 13: Taraxa-evm/core/vm/evm.go (Lines 295,297)

```
291 // initialise a new contract and set the code that is to be used
 ↳ by the
292     // EVM. The contract is a scoped environment for this
 ↳ execution context
293     // only.
294     contract := NewContract(CallFrame{caller, new_acc, nil, gas,
 ↳ value}, code)
295     ret, err = self.run(&contract, false)
296     // check whether the max code size has been exceeded
297     maxCodeSizeExceeded := self.rules.IsEIP158 && len(ret) >
 ↳ MaxCodeSize
298     // if the contract creation ran successfully and no errors
 ↳ were returned
299     // calculate the gas required to store the code. If the code
 ↳ could not
300     // be stored due to not enough gas set an error and let it be
 ↳ handled
301     // by the error checking condition below.
302     if err == nil && !maxCodeSizeExceeded {
303         createDataGas := uint64(len(ret)) * CreateDataGas
304         if contract.UseGas(createDataGas) {
305             new_acc.SetCode(ret)
```

```
306         } else {
307             err = ErrCodeStoreOutOfGas
308         }
309     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Evaluate the return value before executing any operations on the return values.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by checking the err value returned by certain operations. fd07c57c0861a5e967b8710c2ec610a1601ba374

5.8 (HAL-11) IMPLICIT MEMORY ALIASING IN LOOP - INFORMATIONAL

Description:

The ApplyDAOHardFork function contains a for loop in which the memory of the addr loop variable is accessed. At each iteration, the value of the next element in the range expression was assigned to the iteration variable; addr did not change, it just changes the value. Therefore, the expression &v referred to the same location in memory.

Code Location:

- Taraxa-evm/taraxa/state/dpos/precompiled/dpos_contract.go
- Taraxa-evm/consensus/misc/dao.go

Listing 14: Taraxa-evm/consensus/misc/dao.go (Line 31)

```
27 func ApplyDAOHardFork(db vm.State) {
28     // Move every DAO account and extra-balance account funds into
↳  the refund contract
29     refund_acc := db.GetAccount(&DAORefundContract)
30     for _, addr := range DAODrainList() {
31         acc := db.GetAccount(&addr)
32         bal := acc.GetBalance()
33         refund_acc.AddBalance(bal)
34         acc.SubBalance(bal)
35     }
36 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Index the ranged map. This takes the address of the actual element at the i-th position, rather than the iteration variable.

Listing 15: Recommended example

```
drainlist = DAODrainList()
for i := range drainlist {
        acc := db.GetAccount(&drainlist[i])
        bal := acc.GetBalance()
        refund_acc.AddBalance(bal)
        acc.SubBalance(bal)
    }
```

Remediation Plan:

PARTIALLY SOLVED: The Taraxa team partially solved the issue by removing the Taraxa-evm/consensus/misc/dao.go file, but there were still instances of this issue in Taraxa-evm/taraxa/state/dpos/precompiled/dpos_contract.go.

5.9 (HAL-12) LACK OF DEFAULT CLAUSE ON SWITCH STATEMENT - INFORMATIONAL

Description:

The lack of the Default clause in a Switch statement cannot be considered as a security flaw itself. However, using non-standard practices in the code could lead to unhandled errors, making the code unsafe. See the Lack of Return Error issue.

Code Location:

- Taraxa-evm/accounts/abi/abi.go

Listing 16: Taraxa-evm/accounts/abi/abi.go (Lines 110,111,116,123)

```
92 // UnmarshalJSON implements json.Unmarshaler interface
93 func (abi *ABI) UnmarshalJSON(data []byte) error {
94     var fields []struct {
95         Type      string
96         Name      string
97         Constant  bool
98         Anonymous bool
99         Inputs    []Argument
100        Outputs   []Argument
101    }
102
103    if err := json.Unmarshal(data, &fields); err != nil {
104        return err
105    }
106
107    abi.Methods = make(map[string]Method)
108    abi.Events = make(map[string]Event)
109    for _, field := range fields {
110        switch field.Type {
111        case "constructor":
112            abi.Constructor = Method{
113                Inputs: field.Inputs,
```

```
114             }
115         // empty defaults to function according to the abi spec
116         case "function", "":
117             abi.Methods[field.Name] = Method{
118                 Name:    field.Name,
119                 Const:   field.Constant,
120                 Inputs:  field.Inputs,
121                 Outputs: field.Outputs,
122             }
123         case "event":
124             abi.Events[field.Name] = Event{
125                 Name:      field.Name,
126                 Anonymous: field.Anonymous,
127                 Inputs:    field.Inputs,
128             }
129         }
130     }
131
132     return nil
```

-
- Taraxa-evm/accounts/abi/argument.go:unpack
- Taraxa-evm/accounts/abi/reflect.go:reflectIntKindAndType
- Taraxa-evm/common/hexutil/hexutil.go:mapError
- Taraxa-evm/vm/evm.go:run
- Taraxa-evm/vm/opcodes.go:IsPush
- Taraxa-evm/rlp/decode.go:wrapStreamError
- Taraxa-evm/rlp/decode.go:decodeByteArray
- Taraxa-evm/rlp/raw.go:readSize
- Taraxa-evm/taraxa/state/dpos/precompiled/dpos_contract.go:RequiredGas
- Taraxa-evm/taraxa/state/dpos/precompiled/dpos_contract.go:Run
- Taraxa-evm/taraxa/trie/writer.go:commit
- Taraxa-evm/taraxa/trie/writer.go:mpt_insert
- Taraxa-evm/taraxa/trie/writer.go:mpt_del
- Taraxa-evm/taraxa/util/util.go:IsReallyNil
- Taraxa-evm/taraxa/util/bin/index.go:DEC_b_endian_compact_64

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Include the default clause to cover the Switch statement with all options, catching unexpected behaviors that could be triggered in the function.

Remediation plan:

SOLVED: The Taraxa team solved several files affected by this issue in commits e634895136bccf651e7dbf1cf965c88f14238752 and ae19c9931b1dc7f8e8eb1e1a8ab313bb6b853caa

5.10 (HAL-13) INSECURE RANDOM NUMBER GENERATOR - INFORMATIONAL

Description:

The rand_uintptr function was making use of the math/rand package to generate unsigned random integers. This library is considered insecure due to weak random number generation.

Code Location:

- Taraxa-evm/taraxa/state/state_evm/addr_hasher.go

Listing 17: Taraxa-evm/taraxa/state/state_evm/addr_hasher.go (Line 20)

```
19 func rand_uintptr() uintptr {
20     return uintptr(rand.Uint64() % uint64(^uintptr(0)))
21 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use the crypto/rand package instead.

Remediation Plan:

SOLVED: The Taraxa team solved the issue by changing the package used to generate random integers. 79df69c0eba79f12ddb83217a1c981f692fccfd3

5.11 (HAL-14) COMPARE INSTEAD OF EQUAL - INFORMATIONAL

Description:

The Get and Put functions were making use of bytes.Compare to determine if the two byte arrays are equal. bytes.Compare returns an integer value based on the number of differences between the two arrays.

Code Location:

- Taraxa-evm/core/vm/contracts.go

Listing 18: Taraxa-evm/core/vm/contracts.go (Line 55)

```
49 func (self *Precompiles) Get(address *common.Address) (ret
↳ PrecompiledContract) {
50     last_byte := address[common.AddressLength-1]
51     if last_byte == 0 {
52         return
53     }
54     ret = self[last_byte-1]
55     if ret != nil && bytes.Compare(address[:common.AddressLength
↳ -1], PrecompiledContractAddrPrefix) != 0 {
56         ret = nil
57     }
58     return
59 }
```

Listing 19: Taraxa-evm/core/vm/contracts.go (Line 62)

```
61 func (self *Precompiles) Put(address *common.Address, contract
↳ PrecompiledContract) {
62     asserts.Holds(bytes.Compare(address[:common.AddressLength-1],
↳ PrecompiledContractAddrPrefix) == 0)
63     last_addr_byte := address[common.AddressLength-1]
64     asserts.Holds(last_addr_byte != 0)
65     pos := last_addr_byte - 1
66     asserts.Holds(self[pos] == nil)
```

```
67     self[pos] = contract
68 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use bytes.Equal instead of determining if two byte arrays are equivalent, as this function only returns true or false.

Listing 20: Example Remediation

```
1 !bytes.Equal(address[:common.AddressLength-1],
↳ PrecompiledContractAddrPrefix)
```

Remediation Plan:

SOLVED: The Taraxa team solved the issue by applying the proposed recommendation. 6cdd71bbd8e653812578b61db0be2e0d6b77197b

5.12 (HAL-15) MULTIPLE TO-DO COMMENTS FOUND ON THE CODE - INFORMATIONAL

Description:

Open To-dos can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Code Location:

Listing 21

```
./core/vm/errors.go:23:// TODO compress to error codes
./core/vm/contract.go:59:// TODO optimize and refactor
./core/vm/evm.go:36:// TODO OF TODOS: migrate away from big.Int to
↳ a fixed u256 library
./core/vm/evm.go:277:    // TODO This should go after the state
↳ snapshot, but this is how it works in ETH
./core/vm/evm.go:282:        // TODO this also should check if new
↳ acc balance is zero, but this is how it works in ETH
./core/vm/evm.go:536:            // TODO preallocate
./rlp/decode.go:133:    // TODO: this could use a Stream from a
↳ pool.
./rlp/decode.go:141:    // TODO: this could use a Stream from a
↳ pool.
./rlp/decode.go:586:        // TODO only if the map is nil?
./taraxa/state/state_db_rocksdb/latest_state.go:142:    self.
↳ writer_thread.Join() // TODO completely async
./taraxa/state/state_evm/account.go:33:// TODO invert
./taraxa/state/dpos/solidity/dpos_contract_interface.sol:108:
↳ // TODO: these 4 methods below can be all replaced by "
↳ getValidator" and "getValidators" calls, but it should be
./taraxa/state/dpos/tests/dpos_test_utils.go:99:// TODO: fix this
./taraxa/state/dpos/precompiled/dpos_contract.go:453:        //
↳ TODO: once we have also votes statistics, use it in calculations
```

```
15 ./taraxa/state/dpos/precompiled/dpos_contract.go:454:        //
↳ TODO: should voter with 1M stake be rewarded same as voter with 10
↳ M stake ???
16 ./taraxa/state/dpos/precompiled/dpos_contract.go:468:    // TODO:
↳ debug check - can be deleted for release
17 ./taraxa/state/dpos/precompiled/dpos_contract.go:624:    // TODO
↳ slashing of balance
18 ./taraxa/state/dpos/precompiled/dpos_contract.go:1012:    // TODO:
↳  measure performance of this call - if it is too bad -> decrease
↳ GetValidatorsMaxCount constant
19 ./taraxa/state/dpos/precompiled/dpos_contract.go:1050:    // TODO:
↳  measure performance of this call - if it is too bad -> decrease
↳ GetValidatorsMaxCount constant
20 ./taraxa/state/dpos/precompiled/dpos_contract.go:1051:    // TODO:
↳  this will be super expensice call probably
21 ./taraxa/state/dpos/precompiled/undelegations.go:18:    // TODO:
↳ we will needed it for slashing
22 ./taraxa/state/state_db/db.go:22:// TODO a wrapper with common
↳ functionality. Delegate only the most low-level stuff to these
↳ interfaces
23 ./taraxa/state/state_transition/state_transition.go:161:    util.
↳ PanicIfNotNil(self.state.Commit(state_root)) // TODO move out of
↳ here, this should be async
24 ./taraxa/state/chain_config/chain_config.go:14:    // TODO: once
↳ we fix tests, this flag can be deleted as rewards should be
↳ processed only in dpos contract
25 ./taraxa/state/internal/coin_trx_perftest/main.go:45:// TODO more
↳ workload profiles
26 ./taraxa/state/api.go:33:    // TODO have single "perm-gen size"
↳ config property to derive all preallocation sizes
27 ./taraxa/trie/writer.go:46:// TODO parallel
28 ./taraxa/trie/writer.go:155:// TODO maybe dirty checking is
↳ worthwhile
29 ./taraxa/trie/writer.go:200:// TODO maybe panic/recover harms
↳ performance
30 ./taraxa/trie/writer.go:277:    // TODO rehash?
31 ./taraxa/trie/reader.go:114:        size, err := rlp.CountValues(
↳ payload) // TODO optimize
32 ./taraxa/trie/reader.go:175:// TODO lazy load? make sure values
↳ are not loaded twice
33 ./taraxa/util/goroutines/sequential_group_executor.go:30:// TODO
↳ use
34 ./taraxa/C/state.go:245:            // TODO: once there is a
↳ stabilized version - remove this flag and use only dpos contract
```

```
35 ./crypto/bn256/google/bn256.go:31:// TODO(agl): keep GF(pš)
↳ elements in Mongomery form.
36 ./crypto/secp256k1/libsecp256k1/src/java/org/bitcoin/
↳ NativeSecp256k1Test.java:14:    //TODO improve comments/add more
↳ tests
37 ./crypto/secp256k1/libsecp256k1/src/java/org/bitcoin/
↳ NativeSecp256k1.java:152:    //TODO add a 'compressed' arg
38 ./crypto/secp256k1/libsecp256k1/src/tests_exhaustive.c:422:    /*
↳ TODO set z = 1, then do num_tests runs with random z values */
39 ./crypto/secp256k1/curve.go:115://TODO: double check if the
↳ function is okay
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider resolving TODOs before deploying the code to a production context. Use a standalone issue tracker or other project management software to keep track of development tasks.

Remediation Plan:

ACKNOWLEDGED: The Taraxa team acknowledged this finding.

5.13 (HAL-16) PANIC IS USED FOR ERROR HANDLING - INFORMATIONAL

Description:

Several instances of the panic function were identified in the codebase. They seem to be used to handle errors. This could cause potential issues, as invoking panic could cause the program to halt executing and crash in some cases. This, in turn, could have a negative impact on the availability of the software to users.

Code Location:

Listing 22

```
./core/vm/memory.go:60:                panic("invalid memory:
↳ store empty")
./core/vm/memory.go:73:         panic("invalid memory: store empty
↳ ")
./common/math/integer.go:61:// MustParseUint64 parses s as an
↳ integer and panics if the string is invalid.
./common/math/integer.go:65:            panic("invalid unsigned 64
↳  bit integer: " + s)
./common/math/big.go:78:// MustParseBig256 parses s as a 256 bit
↳ big integer and panics if the string is invalid.
./common/math/big.go:82:                panic("invalid 256 bit
↳ integer: " + s)
./common/types.go:247:// If b is larger than len(a) it will panic.
./common/hexutil/hexutil.go:74:// MustDecode decodes a hex string
↳ with 0x prefix. It panics for invalid input.
./common/hexutil/hexutil.go:78:         panic(err)
./common/hexutil/hexutil.go:105:// It panics for invalid input.
./common/hexutil/hexutil.go:126:                panic("weird big.
↳ Word size")
./common/hexutil/hexutil.go:162:// It panics for invalid input.
./common/hexutil/hexutil.go:166:                panic(err)
./rlp/encode.go:101:                    panic(rec)
./rlp/encode.go:106:                    panic(err)
./rlp/encode.go:177:    panic("list is not closed")
./rlp/encode.go:193:    panic("list is already closed")
```

```
./rlp/encode.go:214:                    panic("the list is not
↳ closed")
./rlp/decode.go:128:// and may be vulnerable to panics cause by
↳ huge value sizes. If
./accounts/abi/type.go:188:                             Name:
↳ ToCamelCase(c.Name), // reflect.StructOf will panic for any
↳ exported field.
./accounts/abi/pack.go:66:              panic("abi: fatal error")
./accounts/abi/pack.go:80:              panic("abi: fatal error")
./taraxa/state/state_db_rocksdb/db.go:150:
↳                               panic(err)
./taraxa/state/state_db_rocksdb/db.go:191:              panic(err)
./taraxa/state/state_evm/evm_state.go:104:              panic("
↳ Refund counter below zero")
./taraxa/state/dpos/tests/dpos_test_utils.go:142:
↳ func(num types.BlockNum) *big.Int { panic("unexpected") },
./taraxa/state/dpos/tests/dpos_test_utils.go:210:
↳ panic(err)
./taraxa/state/dpos/precompiled/validators.go:92:
↳ panic("ModifyDelegation: validator cannot be nil")
./taraxa/state/dpos/precompiled/validators.go:97:
↳ panic("ModifyValidator: non existent validator")
./taraxa/state/dpos/precompiled/validators.go:158:
↳ panic("ModifyValidatorInfo: validator_info cannot be nil")
./taraxa/state/dpos/precompiled/validators.go:163:
↳ panic("ModifyValidatorInfo: non existent validator")
./taraxa/state/dpos/precompiled/iterable_map.go:39:
↳ panic("Account already exists")
./taraxa/state/dpos/precompiled/iterable_map.go:74:
↳ panic("Unable to delete account " + account.String() + ". No
↳ accounts in iterable map")
./taraxa/state/dpos/precompiled/iterable_map.go:80:
↳ panic("Account does not exist")
./taraxa/state/dpos/precompiled/iterable_map.go:105:
↳ panic("Unable to delete account " + account.String() + ". Account
↳ not found")
./taraxa/state/dpos/precompiled/iterable_map.go:162:
↳                   panic("Unable to find account " + account.
↳ String())
./taraxa/state/dpos/precompiled/delegations.go:58:
↳ panic("ModifyDelegation: delegation cannot be nil")
./taraxa/state/dpos/precompiled/delegations.go:65:
↳ panic("ModifyDelegation: non existent delegation")
```

```
39 ./taraxa/state/dpos/precompiled/dpos_contract.go:438:
↳                           panic("update_rewards - non existent
↳ validator")
40 ./taraxa/state/dpos/precompiled/dpos_contract.go:471:
↳ panic(errorString)
41 ./taraxa/state/dpos/precompiled/dpos_contract.go:903:
↳ panic("registerValidator: delegation already exists")
42 ./taraxa/state/dpos/precompiled/dpos_contract.go:999:
↳ panic("getValidators - unable to fetch validator info data")
43 ./taraxa/state/dpos/precompiled/dpos_contract.go:1023:
↳                   panic("getValidators - unable to fetch validator
↳ data")
44 ./taraxa/state/dpos/precompiled/dpos_contract.go:1029:
↳                   panic("getValidators - unable to fetch validator
↳ info data")
45 ./taraxa/state/dpos/precompiled/dpos_contract.go:1063:
↳                   panic("getDelegatorDelegations - unable to fetch
↳ delegation data")
46 ./taraxa/state/dpos/precompiled/dpos_contract.go:1075:
↳                   panic("getDelegatorDelegations - unable to state
↳ data")
47 ./taraxa/state/dpos/precompiled/dpos_contract.go:1100:
↳                   panic("getUndelegations - unable to fetch
↳ undelegation data")
48 ./taraxa/state/dpos/precompiled/dpos_contract.go:1134:
↳ panic("state_get_and_decrement - unable to fetch undelegation data
↳ ")
49 ./taraxa/state/dpos/precompiled/dpos_contract.go:1157:
↳ panic("apply_genesis_entry: state already exists")
50 ./taraxa/state/dpos/precompiled/dpos_contract.go:1161:
↳ panic("apply_genesis_entry: owner already exists")
51 ./taraxa/state/dpos/precompiled/dpos_contract.go:1171:
↳ panic("apply_genesis_entry: validator does not exist")
52 ./taraxa/state/dpos/precompiled/dpos_contract.go:1178:
↳                   panic("apply_genesis_entry: delegation is lower
↳ then the minimum")
53 ./taraxa/state/dpos/precompiled/dpos_contract.go:1181:
↳                   panic("apply_genesis_entry: delegation is bigger
↳ then the maximum")
54 ./taraxa/state/dpos/precompiled/dpos_contract.go:1184:
↳                   panic("apply_genesis_entry: validator delegation
↳ exceeds ValidatorMaximumStake")
55 ./taraxa/state/dpos/precompiled/dpos_contract.go:1197:
↳                   panic("apply_genesis_entry: broken state")
```

```
./taraxa/state/dpos/precompiled/dpos_contract.go:1236:// Safe add64, that panics on overflow (should never happen - misconfiguration)
./taraxa/state/dpos/precompiled/dpos_contract.go:1240: panic("addition overflow " + strconv.FormatUint(a, 10) + " " + strconv.FormatUint(b, 10))
./taraxa/state/state_db/db.go:33:                panic(ErrFutureBlock(fmt.Sprint("Requested blk num:", blk_n, ", last committed:", last_committed_blk_n)))
./taraxa/state/state_transition/state_transition.go:134:                                panic("Stats rewards enabled but no dpos contract registered")
./taraxa/state/internal/coin_trx_perftest/main.go:116:                func(num types.BlockNum) *big.Int { panic("unexpected") },
./taraxa/trie/encoding.go:86:            panic("can't convert hex key of odd length")
./taraxa/trie/writer.go:129:    panic("impossible")
./taraxa/trie/writer.go:148:                    panic(issue)
./taraxa/trie/writer.go:197:    panic("impossible")
./taraxa/trie/writer.go:200:// TODO maybe panic/recover harms performance
./taraxa/trie/writer.go:208:                    panic(mpt_del_not_found)
./taraxa/trie/writer.go:255:            panic(mpt_del_not_found)
./taraxa/trie/writer.go:257:    panic("impossible")
./taraxa/trie/layout.go:40:func (self value_node) get_hash() *node_hash { panic("N/A") }
./taraxa/trie/reader.go:61:             panic("impossible")
./taraxa/trie/reader.go:96:             panic("impossible")
./taraxa/trie/reader.go:122:                    panic("impossible")
./taraxa/trie/reader.go:131:                    panic("impossible")
./taraxa/trie/reader.go:134:            panic("impossible")
./taraxa/util/util.go:51:               panic(value)
./taraxa/util/asserts/index.go:10:                      panic(fmt.Sprint(a, " != ", b))
./taraxa/util/asserts/index.go:18:                              panic("assertion error")
./taraxa/util/asserts/index.go:20:              panic(strings.Join(msg, " "))
./taraxa/util/keccak256/keccak256.go:81:                panic("already initialized: either by you or lazily")
```

```
./taraxa/util/bin/index.go:122: panic("impossible")
./taraxa/util/bin/index.go:209:         panic("Could not determine native endianness.")
./taraxa/C/state.go:228:                panic(errorString)
./crypto/bn256/bn256_fuzz.go:31:                panic("parse mismatch")
./crypto/bn256/bn256_fuzz.go:43:                panic("parse mismatch")
./crypto/bn256/bn256_fuzz.go:55:                panic("add mismatch")
./crypto/bn256/bn256_fuzz.go:75:                panic("parse mismatch")
./crypto/bn256/bn256_fuzz.go:87:                panic("scalar mul mismatch")
./crypto/bn256/bn256_fuzz.go:105:               panic("parse mismatch")
./crypto/bn256/bn256_fuzz.go:117:               panic("parse mismatch")
./crypto/bn256/bn256_fuzz.go:123:               panic("pair mismatch")
./crypto/secp256k1/panic_cb.go:11:// recoverable Go panics.
./crypto/secp256k1/panic_cb.go:15:      panic("illegal argument: " + C.GoString(msg))
./crypto/secp256k1/panic_cb.go:20:      panic("internal error: " + C.GoString(msg))
./crypto/secp256k1/curve.go:245:                panic("can't handle scalars > 256 bits")
./crypto/secp256k1/secp256.go:154:              panic("libsecp256k1 error")
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Instead of using panics, custom errors should be defined and handled accordingly, rather than halting the EVM.

Remediation Plan:

ACKNOWLEDGED: The Taraxa team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN