Go Ethereum Security Audit

Proof Of Memes Foundation

Executive Summary

ContractChecker has received an application on October 20, 2022 and performed a security review of the Go-Ethereum implementation of Proof Of Memes Foundation. After a detailed manual review and several tests conducted: ContractChecker verifies the code is written with high quality and developed with a security-focused mindset. No critical security vulnerabilities have been found during advanced analysis and manual reviews.

Smart Chain Explorer:	https://memescan.io/
Web Site:	https://www.proofofmemes.org/
Documentation:	https://docs.memescan.io/smart-chain/guides/concepts/genesis.html
Audited Codebase:	https://github.com/pomnetwork

Contents

1 Purpose and scope

ContractChecker has performed a detailed examination in October 2022 at the request of the Proof of Memes Foundation performed a security review of Go Ethereum, the official Go implementation of Proof Of Memes.

The following components were in primary scope:

- Static analysis
- Package vulnerability analysis
- Peer-to-peer communication and networking, block downloading
- RPC interface, Javascript scripting engine
- Transaction and block processing, consensus rules implementation
- Ethereum Virtual Machine (resource exhaustion/Denial-of-Service)

Specifically not in scope:

- Light client implementation, light client specific syncing
- Backend database, leveldb storage, trie implementation
- Account handling, crypto, handling of private keys and wallets
- Swarm

2 Methods

Advanced static analysis technics conducted to source code along with manual line by line review and forming an understanding of how the application works. More specifically, ContractChecker has:

- Tried to identify possible bottlenecks that can lead to Denial-of-Service (DoS)
- Verified that data shared between threads is properly protected
- Followed the data flow from external inputs to see if invalid input somehow can lead to erroneous behavior
- Performed fuzzing of the RLP serialization format (and the EVM bytecode interpreter, with the existing go-fuzz entry point)

Dynamic analysis has been performed by running a private Ethereum network with two geth nodes and one bootnode, and by writing new unit tests as well as modifying existing tests.

3 Results

3.1 Static Analysis Result

3.1.1 Error Unhandled

There are various number of unhandled errors in Go-Ethereum directory which are low level severity and if any error returned should be managed to reduce the risk of fail. They can be mitigated as below explained example as well.

Example:

func (p TestStruct) Write(p string) error {
…}

Use of Write:

p.Write(s)

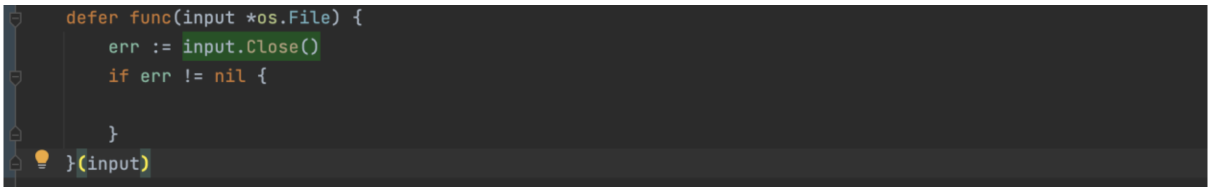
Mitigation: Error unhandled issues might be mitigated as below

Err := p.Write(s)
If(Err != nil) { Return Err}

3.1.2 Deferring unsafe method "Close" on type "*os.File"

There are various number of "deffering unsafe method "close" on type "*os.file" issue in Go-Ethereum directory which are low level severity issues but if any error returned should be managed to reduce the risk of fail. They can be mitigated as below explained example as well.

Mitigation: It is recommended to define a function that contains call method close with the error handled.



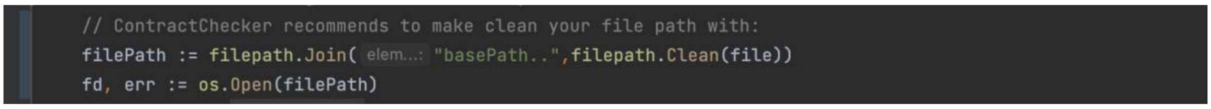
```
defer func(input *os.File) {
    err := input.Close()
    if err != nil {

    }
}(input)
```

3.1.3 Potential file inclusion via variable

There are various number of "potential file inclusion via variable" issue in Go-Ethereum directory which are low level severity issues but if any error returned should be managed to reduce the risk of fail. They can be mitigated as below explained example as well.

Mitigation: Below method is recommended instead of using in argument as file path.



```
// ContractChecker recommends to make clean your file path with:
filePath := filepath.Join( elem...: "basePath..",filepath.Clean(file))
fd, err := os.Open(filePath)
```

3.1.4 Use of week random number generator

There are various number of "use of week random number generator" issue in Go-Ethereum directory which are low level severity issues but if any error returned should be managed to reduce the risk of fail. They can be mitigated as below explained example as well.

Mitigation: It is recommended to use crypto/rand instead of math/rand

3.2 Package Vulnerability Analysis

There was not found any issue in 49 packages scanned against any suspicious, anormal and/or useless code.

Test Item	Status	Issues found
pkg:golang/github.com/common-nighthawk/go-figure@v0.0.0-20200609044655-c4b36f998cf2	PASS	0
pkg:golang/github.com/Masterminds/vcs@v1.13.1	PASS	0
pkg:golang/github.com/Masterminds/semver@v0.0.0-20190925130524-317e8cce5480	PASS	0
pkg:golang/github.com/armon/go-radix@v0.0.0-20180808171621-7fddfc383310	PASS	0
pkg:golang/github.com/boltdb/bolt@v1.3.1	PASS	0
pkg:golang/google.golang.org/protobuf@v1.25.0	PASS	0
pkg:golang/github.com/golang/protobuf@v1	PASS	0
pkg:golang/github.com/golang/protobuf@v1.4.1	PASS	0
pkg:golang/github.com/golang/dep@v0.5.4	PASS	0
pkg:golang/github.com/pkg/errors@v0.9.1	PASS	0
pkg:golang/github.com/jmank88/nuts@v0.4.0	PASS	0
pkg:golang/github.com/nightlyone/lockfile@v1.0.0	PASS	0
pkg:golang/github.com/sdboyer/constext@v0.0.0-20170321163424-836a144573533	PASS	0
pkg:golang/golang.org/x/sync@v0.0.0-20190423024810-112230192c58	PASS	0
pkg:golang/golang.org/x/sys@v0.0.0-20220722155257-8c9f86f7a55f	PASS	0
pkg:golang/github.com/pelletier/go-toml@v1.2.0	PASS	0
pkg:golang/github.com/mitchellh/go-homedir@v1.1.0	PASS	0
pkg:golang/github.com/blang/semver@v3.5.1	PASS	0
pkg:golang/github.com/google/go-querystring@v1.0.0	PASS	0
pkg:golang/golang.org/x/crypto@v0.0.0-20201221181555-eec23a3978ad	PASS	0
pkg:golang/github.com/google/go-github/v30@v30.1.0	PASS	0
pkg:golang/github.com/inconshreveable/go-update@v0.0.0-20160112193335-8152e7eb6ccf	PASS	0
pkg:golang/github.com/tcnksm/go-gitconfig@v0.1.2	PASS	0
pkg:golang/github.com/ulikunitz/xz@v0.5.	PASS	0
pkg:golang/golang.org/x/net@v0.0.0-20190620200207-3b0461eec859	PASS	0
pkg:golang/golang.org/x/oauth2@v0.0.0-20190604053449-0f29369cfe45	PASS	0
pkg:golang/github.com/rhysd/go-github-selfupdate@v1.2.3	PASS	0
pkg:golang/github.com/sirupsen/logrus@v1.6.0	PASS	0
pkg:golang/github.com/sonatype-nexus-community/go-sona-types@v0.1.6	PASS	0
pkg:golang/github.com/shopspring/decimal@v1.2.0	PASS	0
pkg:golang/github.com/go-yaml/yaml@v2.3.0	PASS	0
pkg:golang/github.com/package-url/packageurl-go@v0.1.0	PASS	0
pkg:golang/github.com/recoilme/pudge@v1.0.3	PASS	0
pkg:golang/github.com/mattn/go-runewidth@v0.0.9	PASS	0
pkg:golang/github.com/jedib0t/go-pretty/v6@v6.0.4	PASS	0
pkg:golang/github.com/logrusorgru/aurora@v2.0.3	PASS	0
pkg:golang/github.com/go-yaml/yaml@v3.0.0-20200313102051-9f266ea9e77c	PASS	0
pkg:golang/github.com/spf13/pflag@v1.0.5	PASS	0
pkg:golang/github.com/spf13/cobra@v1.0.0	PASS	0
pkg:golang/github.com/fsnotify/fsnotify@v1.4.7	PASS	0
pkg:golang/github.com/hashicorp/hcl@v1.0.0	PASS	0
pkg:golang/github.com/magiconair/properties@v1.8.1	PASS	0
pkg:golang/github.com/mitchellh/mapstructure@v1.1.2	PASS	0
pkg:golang/github.com/spf13/afero@v1.1.2	PASS	0
pkg:golang/golang.org/x/text@v0.3.8	PASS	0
pkg:golang/github.com/spf13/cast@v1.3.0	PASS	0
pkg:golang/github.com/spf13/jwalterweatherman@v1.0.0	PASS	0
pkg:golang/github.com/subosito/gotenv@v1.2.0	PASS	0
pkg:golang/github.com/go-ini/ini@v1.60.1	PASS	0
pkg:golang/github.com/spf13/viper@v1.7.1	PASS	0

3.3 Peer-to-peer (p2p) and networking

ContractChecker has conducted a crash test with Go-fuzz for RLP and manually reviewed the p2p and networking code, focusing on:

Test Item	Status	Issues found
Secure channel establishment - handshake and establishment of shared secrets	PASS	0
Secure channel properties - confidentiality and integrity	PASS	0
Message serialization	PASS	0
Node discovery	PASS	0
Protection against Denial-of-Service: timeouts and message size limits	PASS	0

3.4 Transaction and block processing

ContractChecker reviewed the transaction and block downloading and processing parts, focusing on:

Test Item	Status	Issues found
Denial-of-Service by memory allocation, goroutine leaks and I/O operations	PASS	0
Synchronization problems	PASS	0

3.5 IPC and RPC interfaces

ContractChecker reviewed the IPC and RPC (HTTP and WS) interfaces, with a focus on:

Test Item	Status	Issues found
Potential access control issues	PASS	0
Request JSON-RPC	PASS	0
Communication IPC (Inter processes communication)	PASS	0

3.6 EVM implementation

ContractChecker reviewed the Ethereum Virtual Machine (EVM) with focus on Denial-of-Service by abusing memory allocation and I/O usage. ContractChecker verified its functionality and couldn't find any issues during fuzzing.

3.6.1 Many third-party dependencies

Go Ethereum depends on approximately hundred third-party packages.

Since each dependency can introduce new attack vectors and requires time and effort to monitor for security vulnerabilities, ContractChecker always recommends that the number of third-party dependencies be kept at a minimum.

Having that many dependencies seem like a lot to handle for any project. ContractChecker recommends that the Proof of Memes developers to make sure all dependencies are really necessary to use, or if some of them can be replaced.