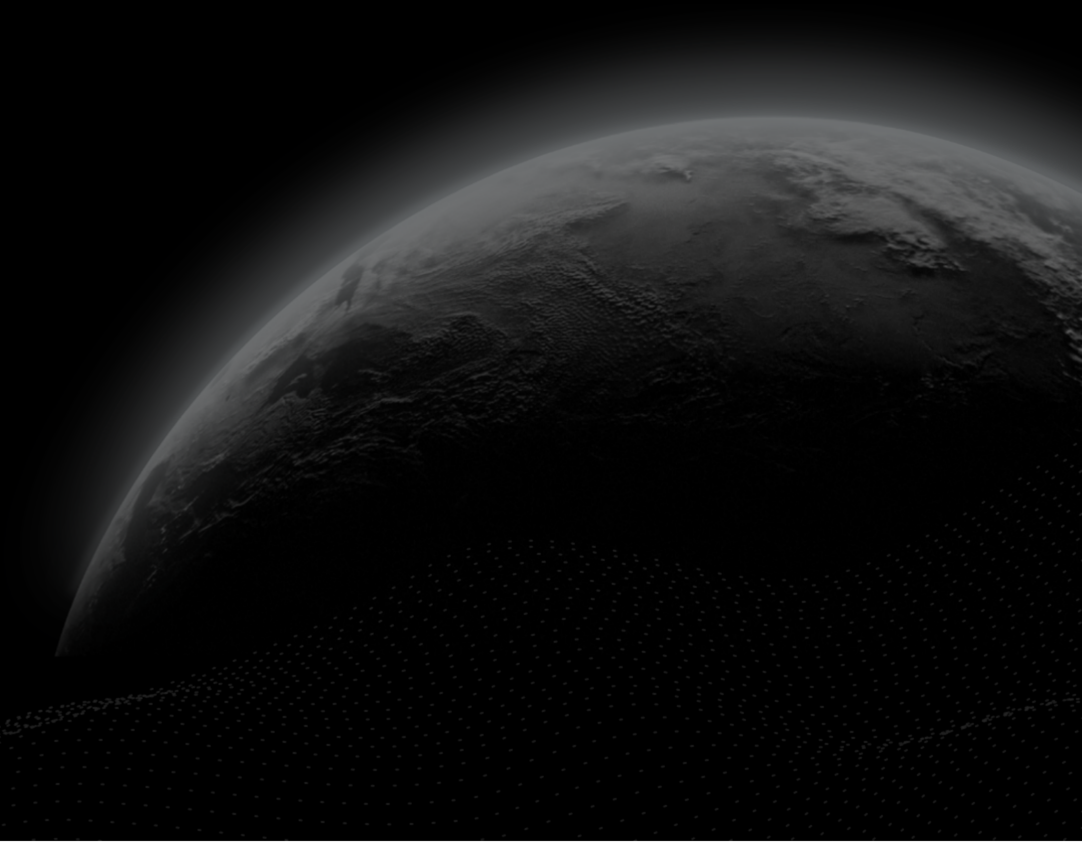
CERTIK

Security Assessment

Bahamut Execution and Consensus

CertiK Assessed on Aug 1st, 2023



CertiK Assessed on Aug 1st, 2023

Bahamut Execution and Consensus

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Chain, Chain-Consensus	Ethereum (ETH)	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Golang, Solidity	Delivered on 08/01/2023	N/A

CODEBASE

- https://github.com/fastexlabs/bahamut-execution
- https://github.com/fastexlabs/bahamut-consensus
- https://github.com/fasttoken1/fasttoken-distribution-eth-

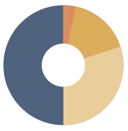
View All in Codebase Page

COMMITS

- af75d5f6c6ab5a33f6a1ac86c5c443e7be943cf1
- 33b75d4e162179d360e60ac88bb4289293b530a6
- 1f2392be6927c2227a0061a5c7c9f7c937545971

View All in Codebase Page

Vulnerability Summary



30	25	0	0	5	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	1	Major	1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨	5	Medium	5 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	9	Minor	6 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	15	Informational	14 Resolved, 1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BAHAMUT EXECUTION AND CONSENSUS

CODEBASE | BAHAMUT EXECUTION AND CONSENSUS

Repository

- https://github.com/fastexlabs/bahamut-execution
- https://github.com/fastexlabs/bahamut-consensus
- https://github.com/fasttoken1/fasttoken-distribution-eth-contracts/tree/master/bahamut

Commit

- af75d5f6c6ab5a33f6a1ac86c5c443e7be943cf1
- 33b75d4e162179d360e60ac88bb4289293b530a6
- 1f2392be6927c2227a0061a5c7c9f7c937545971
- 3226f8330911cb8df77e775f0155b335ba771bd8
- cffbd04e743737989e44cf0ebae70fd353c5a539
- 716ea69939139eab9f45b4c68347eb67de492bea
- b46a400918dd7993f67ac81b8b06a010173a9d67

AUDIT SCOPE | BAHAMUT EXECUTION AND CONSENSUS

180 files audited ● 1 file with Acknowledged findings ● 17 files with Resolved findings ● 162 files without findings

ID	Repo	File	SHA256 Checksum
● FTN	fasttoken1/fasttoken-distribution-eth-contracts	bahamut/FTNVault.sol	8ccc4b1a0687a2919b0315fc77c428b16743ccfc8f96261a561353c8f414ae64
● ACT	fastexlabs/fastexchain-consensus	beacon-chain/execution/activities_processing.go	6ebea7378ccc959d1cef61cc0704ce3eed8258cac82a7d97fd98c7fa24da6ba2
● NOD	fastexlabs/fastexchain-consensus	beacon-chain/node/node.go	3ad29f7da17f4546b190a1d263ff8d39925566769c790f31f93c704842e9bcaf
● STR	fastexlabs/fastexchain-consensus	beacon-chain/rpc/apimiddleware/structs.go	632ee6f9d1a1582465231b9d92dd0ffaa1079b729478431c635f4b077327cae8
● ACV	fastexlabs/fastexchain-consensus	beacon-chain/core/blocks/activities.go	b9667613e95bafbe79fdf15f64607cbb0c6f15fb9254984a332b5c2ccdce63e2
● SYN	fastexlabs/fastexchain-consensus	beacon-chain/core/altair/sync_committee.go	36bf335748e4dc3c581cb47097fa4574eaf5adc9aa82ae1d217479b2f131e22e
● VAT	fastexlabs/fastexchain-consensus	beacon-chain/core/helpers/validators.go	4411a07b67fb7fbc27d42eec9901c232709c218a8143e9bd37dd402f32f04829
● ATT	fastexlabs/fastexchain-consensus	beacon-chain/core/fastex-phase1/attestation.go	f192c3d93d83a168d384083a202817c101b5daefa192b8f98e30ab42a6dce01f
● REW	fastexlabs/fastexchain-consensus	beacon-chain/core/fastex-phase1/reward.go	40aa46c32d5fda20cdc0517bc3a8939f1ddb21a363870f14565f08ac9f024ba2
● FLA	fastexlabs/fastexchain-consensus	config/features/flags.go	b7c8e7f728b17a9d292876a9bd63c834835ed47345c924ce7054ec70d9adaf0e
● CON	fastexlabs/fastexchain-consensus	config/features/config.go	7065ecea4c87a4b5fd304d2c4d35fdca200c5a5c23192134f8a8fef7dfb6b165
● COF	fastexlabs/fastexchain-consensus	config/params/config.go	02f9a809de8d45c974b0536254451e3c0d2e1e4991ab5b938a8ae96011362a91

ID	Repo	File	SHA256 Checksum
● MAI	fastexlabs/fastexchain-consensus	config/params/mainnet_config.go	700b04a4a213e41b43b546e807a259ffc51396fec3553105c274dd7dfb68fc22
● WEB	fastexlabs/fastexchain-consensus	validator/keymanager/remote-web3signer/v1/web3signer_types.go	413009e417bc42ccaa8d76a8d4a79de6e698e34aa62c1826fb4ff5592b287c4d
● CUS	fastexlabs/fastexchain-consensus	validator/keymanager/remote-web3signer/v1/custom_mappers.go	9799aec08355d1545d470b29cbf43a110be886e34225c8075ebaad3145b3f47e
● EVM	fastexlabs/fastexchain-execution	core/vm/evm.go	9a46fa74670d2380eb8f9050ea2919fc3998438744825a8cd7757424cdb4e641
● STE	fastexlabs/fastexchain-execution	core/state/state_object.go	bfa92f906a29579f32cb7a711896feb45ce7119e2105c6456fc66fdf8fb1b995
● STF	fastexlabs/fastexchain-execution	core/vm/stateful_contracts.go	84a5da62ef44b1b4f9a314bfe910cd1ba6a1174e5dfcdf0225af68bc639829d7
○ SER	fastexlabs/fastexchain-consensus	beacon-chain/execution/service.go	151dda0e8f4b337e4a554da01820e4e649eefd5261864fff7221d7c9e110433d
○ OPT	fastexlabs/fastexchain-consensus	beacon-chain/execution/options.go	0cd9c3d5b966b651c09e3f4af8188a4714891acc89278529ed0221d5ac2bcc47
○ LOG	fastexlabs/fastexchain-consensus	beacon-chain/execution/log_processing.go	b21e1f58dfe1c76561e18cd18c29eaaff956a2daf4585ec2bb1543854153ef15
○ CHE	fastexlabs/fastexchain-consensus	beacon-chain/execution/check_transition_config.go	4484c25effe87945d7bba007f26dac64edb1fbec031a1d88efc8cf0d51b84d5f
○ SEV	fastexlabs/fastexchain-consensus	beacon-chain/p2p/service.go	e1a0f5a9ea64286db3f926b9ca38891794307a74b1ba63c648769d716125aaa0
○ OPI	fastexlabs/fastexchain-consensus	beacon-chain/p2p/options.go	03871de7ba8ef05f58cca40bd6016e1714a799bcd26d67aa2cabd80370f46561
○ PUB	fastexlabs/fastexchain-consensus	beacon-chain/p2p/pubsub.go	0c62160d869dd24759078e65cff9a781266b9ddb749508016011e21bf0f87275
○ UTI	fastexlabs/fastexchain-consensus	beacon-chain/p2p/utils.go	48a3b12b7b2400e9fd5fc9773b2b8013fba5ac960622a1fb752016c83888de10

ID	Repo	File	SHA256 Checksum
● EXE	fastexlabs/fastexchain-consensus	beacon-chain/db/kv/execution_chain.go	83e44958a77fe478f86ccbd4d6d858eda6ca8d56dfc752528ed76af9ef4b72ec
● FIN	fastexlabs/fastexchain-consensus	beacon-chain/db/kv/finalized_block_roots.go	74ce8cd3facdac138b7d0526cc04ec6fda50251d8ff9aae10065ae4c2b422e67
● GEN	fastexlabs/fastexchain-consensus	beacon-chain/db/kv/genesis.go	8c6ca04f11e56c0dfdae0b349767b7a1cd549e12ee34174085d33ec39ea4018f
● STA	fastexlabs/fastexchain-consensus	beacon-chain/db/kv/state.go	6c0aeb9e82bd9d8954831ad74dbf2a387c256b13a1274ca523bdc311f950753a
● BLO	fastexlabs/fastexchain-consensus	beacon-chain/db/kv/blocks.go	34b6e0429e89865b73d81f79b36f4bfc2250e473bdf71046eace76da4c6e30e7
● PEN	fastexlabs/fastexchain-consensus	beacon-chain/sync/pending_blocks_queue.go	17d0d84fea6b679f3e085de5d3eddbb01016a42f391cc90938fdb2f73b9a96f0
● MET	fastexlabs/fastexchain-consensus	beacon-chain/sync/metrics.go	a8f5c0d119186fe04a47f739d5292a6ff6be1be710b0ff45dd28c2dcae6c06d3
● RPC	fastexlabs/fastexchain-consensus	beacon-chain/sync/rpc_beacon_blocks_by_range.go	b80d4d312e96f1954c827953a2deb9ed7d51ab397a18d70807a77c7079eecb8c
● VAL	fastexlabs/fastexchain-consensus	beacon-chain/sync/validate_beacon_blocks.go	5e425132fbb70d71d7c3277e550dc3de197f640c4948e332f047635b19d68a79
● ROU	fastexlabs/fastexchain-consensus	beacon-chain/sync/initial-sync/round_robin.go	e5c3f980a65c802c9e099f69ce173d75f4079fe19db2117ed1e497b2bf7aabea
● BLC	fastexlabs/fastexchain-consensus	beacon-chain/sync/initial-sync/blocks_fetcher_utils.go	d941b911fa79c5dffc9c9d0a949fe3ba6561e17a850d63e831609a15a35cc523
● SEI	fastexlabs/fastexchain-consensus	beacon-chain/rpc/service.go	29ef1b72db1644517aba4914d48513af3289d84feebff33541233f9ece322cef
● FET	fastexlabs/fastexchain-consensus	beacon-chain/rpc/statefetcher/fetcher.go	337c9e155c0e291def7f90047622bdb0e9278c160077b03c384d34417e38d4ce
● VAI	fastexlabs/fastexchain-consensus	beacon-chain/rpc/eth/validator/validator.go	ed349ba6bcf5fc457a73788f61428a04f8b3d4b0537b5683aa63f6196a6bc518
● VAD	fastexlabs/fastexchain-consensus	beacon-chain/rpc/eth/beacon/validator.go	10e30f88cc4d8adbce6208af6b6ddc974a420196d052d7f232de0eb909087452

ID	Repo	File	SHA256 Checksum
● BLK	fastexlabs/fastexchain-consensus	beacon-chain/rpc/eth/beacon/blocks.go	bcc3981ae78a63b05b36b0d89b912260ffaf9e03b531d5c8202a089fc4d10c30
● SEE	fastexlabs/fastexchain-consensus	beacon-chain/rpc/prysm/v1alpha1/validator/server.go	b82ea7ab674e6df8d3a29103324191aaf815c77ffc1ad2d4d6fa25cf32172842
● PRO	fastexlabs/fastexchain-consensus	beacon-chain/rpc/prysm/v1alpha1/validator/proposer_execution_payload.go	c9c7c8823520404097342 97ad70271d146772a8fdb692a2482a06bb134c679b6
● PRP	fastexlabs/fastexchain-consensus	beacon-chain/rpc/prysm/v1alpha1/validator/proposer_altair.go	4bb85988817e8b4c72fadf929da2e98d3e056691dfa5caf2656d6132399f993e
● PRS	fastexlabs/fastexchain-consensus	beacon-chain/rpc/prysm/v1alpha1/validator/proposer.go	5a702c52b1f3d2dcf6bc6ab36a1560956614db2c3fceeb14f7d8ed1a4bc8b147
● PRE	fastexlabs/fastexchain-consensus	beacon-chain/rpc/prysm/v1alpha1/validator/proposer_bellatrix.go	8e79c63f42f1e8a52bf54b9a2ed3b27b039345c76125edfc855c4c89209b7480
● PRR	fastexlabs/fastexchain-consensus	beacon-chain/rpc/prysm/v1alpha1/validator/proposer_activities.go	050c311cd2c39993daf994a2d72f432e9b19693f0ce972aeca238d8ebeff8c32
● BLS	fastexlabs/fastexchain-consensus	beacon-chain/rpc/prysm/v1alpha1/beacon/blocks.go	f08a09f7624816a8537722b2c27622bf26e317e31df5d10b34a6e5ddb1656433
● SEC	fastexlabs/fastexchain-consensus	beacon-chain/deterministic-genesis/service.go	2456dc966d83b3f364ded15642aee91e53e36b259721e543042fdb90a1e1ba70
● PRC	fastexlabs/fastexchain-consensus	beacon-chain/monitor/process_block.go	46142f6a5337bc5de1957610490403f464f565de959833247d8497f6fa921cea
● OPO	fastexlabs/fastexchain-consensus	beacon-chain/builder/option.go	70f970cfaaee8b3c8ba1560eac9d4b9e23b4202d033ffa33492590b5b97be8c0
● SEB	fastexlabs/fastexchain-consensus	beacon-chain/builder/service.go	46c4b52e1addaed45bd595d30db44e14ddde5747d143227220a13c9c6efdcf95
● LOA	fastexlabs/fastexchain-consensus	beacon-chain/cache/activitycache/log.go	cd79e6f7ecd2585fe8300c72847809aefd352410093c8b56fddd9f2010791215
● ACI	fastexlabs/fastexchain-consensus	beacon-chain/cache/activitycache/activity_change_cache.go	234579e41696f2d64c0e4cc1d9c9cb2f14c72d1586b420fa43d24d049410c643

ID	Repo	File	SHA256 Checksum
● INT	fastexlabs/fastexchain-consensus	beacon-chain/state/interfaces.go	5c02e9aff01be7ff05dd2ee82d7d8c20a0d09add0b787d19361e2615d8fd23ae
● FIE	fastexlabs/fastexchain-consensus	beacon-chain/state/fieldtrie/field_trie_helpers.go	05a8042f689df3e2acd72b59b870e80adce4655735129bfa94400c2841d88adc
● REA	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/readonly_contracts.go	f2e22a10a8c0550473bd3b2d5c7bd82288ab4f6bd5182a15179b4d5cfb814700
● GET	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/getters_misc.go	3c21bd3fed4f9a4c69cb1d53fe3eb1834800e22d22562c5d30b88e820b8b30c2
● BEC	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/beacon_state_minimal.go	e1a08feb54adb1d6e595cde4bcea3ac8c97eb17de065b111296e5b8ed450c4e6
● STT	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/state_trie.go	16801728ad6c50d0b8e297bb008663340d8a5ea5239ae7900ed6362a77dc2121
● TYS	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/types.go	59166a677563e7b7e19089e8b9333f3f353bb09fbf06520d92910d40d18b2aee
● SET	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/setters_eth1.go	963505ab4996beae6c13f74eb39978fa133c8a9eb51ea29800aef2ce17efe0f2
● GEE	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/getters_validator.go	78592781fe01eda0a4347c1602aa2aa90dad27a0443b7756c10134e1f54cf38b
● BEO	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/beacon_state_mainnet.go	4c05a86824b19dcce1b72f4f3130da74960980767b58e17aea4c7e88d1fca20f
● SES	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/setters_validator.go	774a34b9a2b3b2a1516b20f38492378581e999b67ec2f207f3c2776fb982d36c
● GER	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/getters_eth1.go	f173d4eb60fc135b22a66259fedb305a6e6cba7303faea6aba3a33672d544689
● RED	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/readonly_validator.go	f2eeb403e604cfbe15e99d0d180e6950dee60629dab28f7fe574e621840ca685
● GES	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/getters_state.go	384b118c475e8f61299fedac2e69bd647cad680c2cfff6cee2d826477a66c8eb
● HAS	fastexlabs/fastexchain-consensus	beacon-chain/state/state-native/hasher.go	c183e1eb287bfc18d785cae56e1782fea91bc90372ada4ae9dd9b17ff13c9ec0

ID	Repo	File	SHA256 Checksum
GEI	fastexlabs/fastexchain-consensus	beacon-chain/state/genesis/genesis.go	006052100fd17a956ce78038f464bb5810dfbe872a213719f43533e4a989d0a3
REP	fastexlabs/fastexchain-consensus	beacon-chain/state/stategen/replay.go	a5f3a08be5bdd1631a68ffe6a827b9d98655728fff825a9d6bbe75700623c10f
HIS	fastexlabs/fastexchain-consensus	beacon-chain/state/stategen/history.go	9cada2e728f3a9fec1005dff57bf6e0c6e2885de484b9fa68cbbbbc5a9fbfa38
GEA	fastexlabs/fastexchain-consensus	beacon-chain/state/stategen/getter.go	e07622cb36d73a7c493072bb0aef5ced8700d899d69efe64bd4940b2bb99b5a8
VAA	fastexlabs/fastexchain-consensus	beacon-chain/state/stateutil/validator_root.go	66efb9446e661b543fc85360f08fbac0f678bc4080817949239d51d2711ce4a6
FIL	fastexlabs/fastexchain-consensus	beacon-chain/state/stateutil/field_root_validator.go	1b06c6858d910a4c0231bbc51bb94e3c9a9e11e435b578c91a3678f65d4032b3
COR	fastexlabs/fastexchain-consensus	beacon-chain/state/stateutil/contracts_root.go	5c6579fe9ae06af3ba3ad7fa52c82ee6ee39c1dac2adcead7486a6e632855d29
FID	fastexlabs/fastexchain-consensus	beacon-chain/state/stateutil/field_root_contracts.go	79f4968208cb3ad9b7a6ba1913f4ae070ef9baead3c8b016a7b2589b2a9d540d
COA	fastexlabs/fastexchain-consensus	beacon-chain/state/stateutil/contracts_map_handler.go	5fa30915814c48c7818bf9ecb9dc19e4c3ad71a67b9083fb1861123061b04cbf
INE	fastexlabs/fastexchain-consensus	beacon-chain/forkchoice/interfaces.go	85e8d2bf3dba91cc2f706fd5a6a86cfd1d526318bf51c7bac0917a9244cb82e4
FOR	fastexlabs/fastexchain-consensus	beacon-chain/forkchoice/doubly-linked-tree/forkchoice.go	ae7d1b8a492e0cb5ccbd935eb0071816a969ec1d5769729746bdd1109349337f
UNR	fastexlabs/fastexchain-consensus	beacon-chain/forkchoice/doubly-linked-tree/unrealized_justification.go	51bd1f1fe4c7987ff6b0bbbe666ee18153908aabba77171452591bdffecdf33b
STO	fastexlabs/fastexchain-consensus	beacon-chain/forkchoice/doubly-linked-tree/store.go	74e21fc79e5783c167dc13669f08d98978db162caf67120cbf82a469adb779b4
TYD	fastexlabs/fastexchain-consensus	beacon-chain/forkchoice/doubly-linked-tree/types.go	5e0cc597e22cae48399850b095eab17901e82b099742be3a031462d0b0b159f5
ONT	fastexlabs/fastexchain-consensus	beacon-chain/forkchoice/doubly-linked-tree/on_tick.go	cd8829feaa28a423ba852aa017ddc54f7bbdf59c30ef24cc9a60cf145003d27a

ID	Repo	File	SHA256 Checksum
NOE	fastexlabs/fastexchain-consensus	beacon-chain/forkchoice/doubly-linked-tree/node.go	c2389bc5d5b247494d16d84607160583da4d6d4b4c7cde184e7ed7f4e8e8d2ce
EXC	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/execution_engine.go	015ee81b4a394bbdd5b8d218631e79d00cd266a4f20ef2e42edd92a8ffe9f303
LOB	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/log.go	590da905bcbd1efb2d708e2e8010235d5c95c26cfc7733efb7b5edacfda2efc3
SEL	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/service.go	eacb2e82aa4449e1b32a5d83b819915cc0944e279e78ba46d70d7f6b96184015
MER	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/metrics.go	9edff79cf0c9cb1d2db9853ce028906ce9c62f511f155b5042e5f9e1d8946bd6
PRB	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/process_block_helpers.go	aee0bf8107153331c8c752254da90da49b31ef44054277f944b08c4eb5885c2a
MEG	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/merge_ascii_art.go	b1452cc13dbc1147574bd723c5cef1b163f5bef60ce887f16e3f7e4e9a588a5d
HED	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/head.go	befd15f3fadd342d9c9fface4b3a06a0150bfac0479d5792579784015739d8d9
PRK	fastexlabs/fastexchain-consensus	beacon-chain/blockchain/process_block.go	e7572d161ba5e77bf85f1c293dd3ae3e9eaf6354d603bb9cc627da12417add56
DES	fastexlabs/fastexchain-consensus	beacon-chain/core/blocks/deposit.go	cb3898c966885a2e0f155cd5f57b3a9ffb6834713c91116c4151bf2784ce4d95
HEE	fastexlabs/fastexchain-consensus	beacon-chain/core/blocks/header.go	26d2c8c83b8598be7008eb4ff9c471842d88803155780758941c47a315ec2717
RAN	fastexlabs/fastexchain-consensus	beacon-chain/core/blocks/randao.go	1ebf807cf39450d5e19746650ffc406fb48dd354dad3248130b31ec4e140959f
SIG	fastexlabs/fastexchain-consensus	beacon-chain/core/blocks/signature.go	15476420e7ef52a4a2f71591f7e9d0e3de6540238800925bc9334b260dd1696a
TRA	fastexlabs/fastexchain-consensus	beacon-chain/core/transition/transition.go	e4fd4d1f29c462ab6a49dd3648e85ae17aaa1f8cfb3b7d9883da1b9eca1c9bec
TRN	fastexlabs/fastexchain-consensus	beacon-chain/core/transition/transition_no_verify_sig.go	fb317dc085f87e7bb0cf222ba3c5749034b837eb2d05a47892c2370de8f4a575

ID	Repo	File	SHA256 Checksum
STN	fastexlabs/fastexchain-consensus	beacon-chain/core/transition/state.go	bcf3ae2fed383d4164039d2b4637cce161c52fcb6f10ac1cea12ee258cdad833
COC	fastexlabs/fastexchain-consensus	beacon-chain/core/transition/stateutils/contracts_index_map.go	80b4968d40c17b6007ae9ba064c3c9fe7c8ea522f1c8435463d3990cc6de9f60
EPO	fastexlabs/fastexchain-consensus	beacon-chain/core/epoch/epoch_processing.go	9604a514eab8ef470f9a1fc5e8a99f68bb44db68073f56ed0f586f548b1fd313
JUS	fastexlabs/fastexchain-consensus	beacon-chain/core/epoch/precompute/justification_finalization.go	c7b7f09e3a69c379170cf40b1d4229e1a374caab944592398f07cfa2b4598355
UPG	fastexlabs/fastexchain-consensus	beacon-chain/core/altair/upgrade.go	192a38e8dbc770a9bd3ef5a0a9c73a296ce95fb13cec223fff7dd4f959b44e77
TRS	fastexlabs/fastexchain-consensus	beacon-chain/core/altair/transition.go	4f31a8bde8643457abc242d7967d69fbb94e13523af7b58cefc0880cb7b94c38
UPR	fastexlabs/fastexchain-consensus	beacon-chain/core/execution/upgrade.go	ab9e7d17c0d912fe6173efc5fbddf620afe06aaf49bfeb7c7ba9ccd5e326feb5
COS	fastexlabs/fastexchain-consensus	beacon-chain/core/helpers/contracts.go	f15254d6d4d400aac8471420284b426a12848b799c0182c12e2398288320a727
BLF	fastexlabs/fastexchain-consensus	beacon-chain/core/fastex-phase1/block.go	407830caab3eda3a516e4ceab7d12c5f6266ff1c248c8a3b551970ff43086d73
EPC	fastexlabs/fastexchain-consensus	beacon-chain/core/fastex-phase1/epoch_precompute.go	a99ff8c140af6c1ebfff832eeaba3dc6f3c8a57951f81673f6ec06941933a798
TRI	fastexlabs/fastexchain-consensus	beacon-chain/core/fastex-phase1/transition.go	78faab36e474697a24ea56b066ea81df8fe506a439caaf60e4213da95e5aae06
UPA	fastexlabs/fastexchain-consensus	beacon-chain/core/fastex-phase1/upgrade.go	e7071a1819233f57aeeb18fb48bb1dcc58345ec38f69f589a895da2aa93b8916
TYP	fastexlabs/fastexchain-consensus	api/client/builder/types.go	1c80a475b41992d46f6a018cec56fd101a47f9c4d5193bc8c464612ee4f4342c
CLI	fastexlabs/fastexchain-consensus	api/client/builder/client.go	90b9471d41d11626c01abf6b11fa48d7e891e20c1033b354a5f236ea052b0365
CHC	fastexlabs/fastexchain-consensus	api/client/beacon/checkpoint.go	bdcbce655bedb3077d4e552c382e06f714d2b932d3dd7b6e3295749910955154

ID	Repo	File	SHA256 Checksum
● ACC	fastexlabs/fastexchain-consensus	cmd/validator/accounts/accounts.go	cd62100204786696a4a4e500c017abb2888c8291fa6f17b401012405140a9317
● IMP	fastexlabs/fastexchain-consensus	cmd/validator/accounts/import.go	b47b1ce285e36920e5d17f2204ac271624c3f24aceb094c2c34d1264aae82810
● CRE	fastexlabs/fastexchain-consensus	cmd/validator/wallet/create.go	52be5f292590c3f2b5fdb7d8eef2e6ecfb7bf20cea249a81563779a7312b548e
● WAL	fastexlabs/fastexchain-consensus	cmd/validator/wallet/wallet.go	dd053f67520c5a2e38520a98d03ebc64c98d4061cfef91ac73b2ea65e8719948
● SLA	fastexlabs/fastexchain-consensus	cmd/validator/slashing-protection/slashing-protection.go	8883abade7034150f3d430ef729bcc06c9a16234963aef72fbbfe5636fcd51e2
● DEP	fastexlabs/fastexchain-consensus	config/features/deprecated_flags.go	703851df0a821069fe32fb43699ca7e8e7071ad0632c5cb0da852178cedb0909
● VAU	fastexlabs/fastexchain-consensus	config/params/values.go	f8853cbf45b3c0f5b3c65bfb4233d45fcdb97829d6cba762d5fd1f5af9817b89
● TES	fastexlabs/fastexchain-consensus	config/params/testnet_fastex_chain_config.go	9a0634ded0fc6fefe33cf1c0bde1bb5a91efa11977d76bbcc619839f1cef3b74
● MIN	fastexlabs/fastexchain-consensus	config/fieldparams/minimal.go	c9cf2513bea3004a9ed335590e22e713f56f1ad27f6b3d1fe7c5b2227a60ab56
● MAN	fastexlabs/fastexchain-consensus	config/fieldparams/mainnet.go	5ffb1b9991670d7e74a7781c1f5a3af77c430d70f42417e237434bbdf0ff828c
● FAC	fastexlabs/fastexchain-consensus	consensus-types/blocks/factory.go	54ad9d69160f092f8a57171ab5c4ce1d410e408e596de8c1defcfdb9ae3cbf8c
● TYE	fastexlabs/fastexchain-consensus	consensus-types/blocks/types.go	09bb36166be78a860087020af15a59536f96cf11c13432dc586590551d16db82
● PRT	fastexlabs/fastexchain-consensus	consensus-types/blocks/proto.go	69e173678993020c4ea5b6883bfc21b7a201e9482172f1a2970f9508b9752cc4
● UTL	fastexlabs/fastexchain-consensus	consensus-types/interfaces/utils.go	be1e99bef31a39519841b7d56bd04e14274c2ee07e6855f1eba3bc71e7bef571
● BEA	fastexlabs/fastexchain-consensus	consensus-types/interfaces/beacon_block.go	086e162bbdc35db34668e8af2301121b56e0e88182c7070e60cf1c5017f843b4

ID	Repo	File	SHA256 Checksum
DEO	fastexlabs/fastexchain-consensus	contracts/deposit/deposit.go	908b4c6c5bac2e7107d0a45b2f3c8e5c6581a37488c00e2c6452a0598c519239
COT	fastexlabs/fastexchain-consensus	contracts/deposit/contract.go	b1b2614a5ceeca140afbc1fd04d37e3a7bc552730baea071818cc91c1dbed027
LOS	fastexlabs/fastexchain-consensus	contracts/deposit/logs.go	b83ec024044c9518235c8021b1ebee3bdaac95755c850f3c0a1afc767cda7d1e
COI	fastexlabs/fastexchain-consensus	encoding/ssz/detect/configfork.go	ecec1ca5ebf3938a59a63bfa9b49d280da7bd4e6818d3fe0723660eb7d958ec9
BYT	fastexlabs/fastexchain-consensus	encoding/bytesutil/bytes.go	f3ae8c06186abc117c6877fc6e6f78301243b9bc8b5ba96f17785747d5bc40f1
JSO	fastexlabs/fastexchain-consensus	proto/engine/v1/json_marshal_unmarshal.go	bcce452afd619e48da828e3cd556f7c4563878a722493366df4cde7243e63697
V1A	fastexlabs/fastexchain-consensus	proto/migration/v1alpha1_to_v1.go	02ca2a4d224027157102458e82c614964cbcabff680f6c2eccf35c941775f0cd
V1L	fastexlabs/fastexchain-consensus	proto/migration/v1alpha1_to_v2.go	9615bc74bf42c8459d19b6c8b304634d17e60dd2df32aa55c44a52691b4f9a66
CLO	fastexlabs/fastexchain-consensus	proto/prysm/v1alpha1/cloners.go	8961903c9d40039f27578c506ee32cf1f732b22e45304d39c3224ac0ef84a0e7
JSN	fastexlabs/fastexchain-consensus	proto/prysm/v1alpha1/json_marshal_unmarshal.go	049a58137e6db84ace1c5e516e7a3bf77bfb4543301f96462544005018142924
MAB	fastexlabs/fastexchain-consensus	tools/benchmark-files-gen/main.go	b09a5ca024e8b3bf4b0f99ecb76ad2f582f74c8f5983613e06c78ed6280abc10
MAL	fastexlabs/fastexchain-consensus	tools/blocktree/main.go	f34eea3513297a61c741ddb30551e0d8fe0d5137be30ded1da6bcf49ab82f0d6
MAP	fastexlabs/fastexchain-consensus	tools/pcli/main.go	5e629096c103f0e5c833a466d21f1df0e23f97dec53ad94f8c6b57665f51e8f6
WAE	fastexlabs/fastexchain-consensus	validator/accounts/wallet_create.go	7cc2f081f5be950480590bb3bb113f6af6f01164e36d6c9bbaeca351ab886311
CLM	fastexlabs/fastexchain-consensus	validator/accounts/cli_manager.go	fdb766c7a35603fd27d521f8557569303e273e1211ada42285f152d160042dd5

ID	Repo	File	SHA256 Checksum
CLP	fastexlabs/fastexchain-consensus	validator/accounts/cli_options.go	b14d53a0a587e3ff517f152996091d12b98f20c9cdcadebae1e4f23a79d3ef27
HEA	fastexlabs/fastexchain-consensus	validator/rpc/health.go	33d4e4f549ac97de017be37865e1dae3ace4f51a41d9fb9d4cf4263435e4446f
WAT	fastexlabs/fastexchain-consensus	validator/rpc/wallet.go	a3a48113d95c57993a0e489e11d156f156e7018b601452ede2ac2025b2a77c50
RUN	fastexlabs/fastexchain-consensus	validator/client/runner.go	760fa70fd6d30377751151ae7debc575aa151d227ac7e9639ab426b05b1c86a5
PRL	fastexlabs/fastexchain-consensus	validator/client/propose.go	e1fb8f4f33f2ea7618a3e0456f4d96e2f19207659792769d061996fa4ab6f8a6
INR	fastexlabs/fastexchain-execution	core/vm/interface.go	49b16b5e29f18bf541b10874071db7ed766939ffdde31cce80dcae72a32fbfb5
OPE	fastexlabs/fastexchain-execution	core/vm/operations_acl.go	2fcab564fa29f2ac2deb3acb7a3dd4d255d8bb85017a93ecf10984bcbd67b25f
COM	fastexlabs/fastexchain-execution	core/vm/contract.go	4ef30570f4452486f1052a64467492311eafa1fcb537be4360a9df0bb7c8addc
STD	fastexlabs/fastexchain-execution	core/state/statedb.go	bd5de7c80e7d9d883ed971f6ef30b6f18065aa7f06b1a6968ecd61dc1f52e9e0
JOU	fastexlabs/fastexchain-execution	core/state/journal.go	a4a8e619777396f51aba6dfdf033a910c3e09da8652e2919dab787f60ad980b1
DUM	fastexlabs/fastexchain-execution	core/state/dump.go	51165d1cba913f26ce17cffec8cd087caf9ac5c02a63aa6c462c8e05c859e8c2
MKA	fastexlabs/fastexchain-execution	core/mkalloc.go	0e9cc0f8ae964c896f27a8e350910b322ecf87a9a054e538d493c8cce7e0bde4
GEL	fastexlabs/fastexchain-execution	core/genesis_alloc.go	f0e28cda91b9a4dc30f796ded46914a46ab90a078d352179b447f162b7b0a232
MAG	fastexlabs/fastexchain-execution	cmd/geth/main.go	a2b4a18385093241a877c9afae0444b37684138f255ebe352889d199284c0e9e
BAC	fastexlabs/fastexchain-execution	eth/backend.go	a07af78235e901f86d0cb5a7777083e791ab841e0e4dc28510ce6e2fb584ae3b

ID	Repo	File	SHA256 Checksum
SYC	fastexlabs/fastexchain-execution	eth/protocols/snap/sync.go	4b5c0f669b3b33c527b756281322f10aecc7a97184fab1a183a8d37896210d82
VER	fastexlabs/fastexchain-execution	params/version.go	76fd56ad95194b7fe575bdba89796891bbc0b4c48740acdaa198405137be4f54
TRE	fastexlabs/fastexchain-execution	trie/trie.go	64c00a91509bf329c0c9b778687814daf8443abf66746ba3fd7b77906af74202
COG	fastexlabs/fastexchain-execution	params/config.go	c2f201bb6944de0cc6240f5bf5a551f9db42776bea95959f4167dfb33997711e
BOO	fastexlabs/fastexchain-execution	params/bootnodes.go	859b6398c5476d1f072bdf959126fcfadf5b1219ae95d73c31be750bb0ef6058
EVV	fastexlabs/fastexchain-execution	core/vm/evm.go	fafb6dfd64906e5ef14e4fd148af460a9808d2674ff2a95cdd41b455c1f3e498
COV	fastexlabs/fastexchain-execution	core/vm/contracts.go	68500457b11c105518225f6b8b501036adb835d13b8ea8dbcba6908356361562
INS	fastexlabs/fastexchain-execution	core/vm/instructions.go	60b2b7ecf929cd77463cbeb6fa7c6b337071c40fde9711999d5bce13147a7ff8
ERR	fastexlabs/fastexchain-execution	core/vm/errors.go	df6cab5ad1e465d61f4ac8b97f958aa4fe6fd0e4d263f6102a3eba7b715ef730
GAS	fastexlabs/fastexchain-execution	core/vm/gas_table.go	252cb027a17fcf081b5afb555e9e25272c483e7dff95007e6da15a641926bfbb
STB	fastexlabs/fastexchain-execution	core/state/statedb.go	296e8da41f5c303e8e3ef71e8c9168c48a299b328fce3b349a74979d7cab11b0
JOR	fastexlabs/fastexchain-execution	core/state/journal.go	6b4ea79cd07ab72c23092e830213ef23d9aa53f36c039edd1f54fe1717e55000
STJ	fastexlabs/fastexchain-execution	core/state/state_object.go	128cc21249d2ec3628bd63f30f0e94a6bc5e689561f5c7d8bf68c1667fe77859
GEP	fastexlabs/fastexchain-execution	core/state/snapshot/generate.go	10ddc38fdcdd973f6ae91b93bbb4bfd7e540cbb0dfb71717f5140d6faab3c640
ACO	fastexlabs/fastexchain-execution	core/state/snapshot/account.go	9edaf7779e8311ee2bda1a8623d925886c20454d1419c26297c1f1d5b24ac3f0

ID	Repo	File	SHA256 Checksum
STS	fastexlabs/fastexchain-execution	core/state_transition.go	e5786e8011c1cf42f04f5523253bad5dcab9faa5ddfcf536e60e1ac4ea1668a6
GEC	fastexlabs/fastexchain-execution	core/genesis.go	a04e5523dc10fbcc38dc21e7dd0f7003128c86f6689f5f2af72f5558da768c4e
SNA	fastexlabs/fastexchain-execution	cmd/geth/snapshot.go	8c2e6fbd530536c7018dd9f3f653a3959a2dd3a18f142eefbe9515bed47861a1
FLG	fastexlabs/fastexchain-execution	cmd/utils/flags.go	31760bb526cbc5c887dd837836d01fcb6b6418c51cf162a37e06ae2ec4f5d71a
API	fastexlabs/fastexchain-execution	internal/ethapi/api.go	37d4e2c77c6f2d1f509e044f60a52a44f3959032312399f559d786f186bcb101

APPROACH & METHODS | BAHAMUT EXECUTION AND CONSENSUS

This report has been prepared for Fasttoken to discover issues and vulnerabilities in the source code of the Bahamut Execution and Consensus project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | BAHAMUT EXECUTION AND CONSENSUS

Overview

The **Bahamut** is built on the Ethereum Proof of Stake, which allows the validators to explicitly stake assets (8192 FTN tokens) in a smart contract as a collateral that will be slashed in the case that the validator behaves dishonestly or does not perform the duty for the consensus.

The validator needs to run three clients: an execution client (Geth), a consensus client (beacon chain), and a validator client. Once the validator is activated after depositing the FTN tokens, the validator will receive new blocks from the peers in the network. The transactions in the block will be re-executed in the execution layer and the signature will be validated to prove the validity of the block. The validator will sign an attestation to vote for the block and gain rewards for successfully participating in the consensus.

The lifecycle of a transaction is illustrated in the following steps:

1. A user submits a transaction to the execution layer via JSON-RPC and will be verified for its validity;
2. If the transaction is valid, then it will be added to the execution layer's mempool and broadcasted to other nodes over the gossip network;
3. Once a node is the block proposer of the current slot which is pre-assigned in a pseudo-random manner with the RANDAO algorithm. The execution layer of the node bundles a batch of transactions from the mempool to create an execution payload, which is passed to the consensus layer to build the beacon block.
4. Other nodes receive the beacon block via the consensus gossip network. The beacon block will be re-executed through the execution layer to ensure the state change is correct.
5. Once the beacon block is validated, the validator client will sign the attestation for the block.
6. A transaction is finalized once it lies in between two checkpoints with a supermajority, that is, two-thirds that the validators can be associated with the contracts that record the total balance of all active validators.

The novelty in the **Bahamut** protocol is that the validators are associated with the contracts that record the gas consumed in the contracts. The gas consumption of the contract owned by the validator is used to define an **activity score** that belongs to the validator, which in turn affects the chance for a validator to be a block proposer as well as the base proposer reward.

The protocols `Bahamut-consensus` and `Bahamut-execution` are forked from `Prysm 3.2.2 & 4.0.3` and `Geth 1.10.26` respectively, in which only the differences between the listed commits are in the audit scope.

Bahamut-consensus:

- https://github.com/fastexlabs/bahamut-consensus/commit/cffbd04e743737989e44cf0ebae70fd353c5a539

Prysm:

- https://github.com/prysmaticlabs/prysm/tree/e2fa7d40e3f496416283cc1d329a8ff6c048cb8a

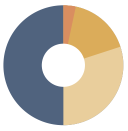
Bahamut-execution:

- https://github.com/fastexlabs/bahamut-execution/commit/716ea69939139eab9f45b4c68347eb67de492bea

Geth:

- https://github.com/ethereum/go-ethereum/tree/e5eb32acee19cc9fca6a03b10283b7484246b15a

FINDINGS | BAHAMUT EXECUTION AND CONSENSUS



30	0	1	5	9	15
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Bahamut Execution and Consensus. Through this audit, we have uncovered 30 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
FTN-04	**Initial Distribution Centralization Risk In Contract `FTNVault`**	**Centralization**	**Major**	● **Acknowledged**
322-01	Missing `Contract` When Processing Deposit Log	Logical Issue	Medium	● Resolved
DEP-02	Potentially Override The Current Owner Of Contract	Logical Issue	Medium	● Resolved
EVM-01	Missing Memory Gas Usage In Activity When Adding It To StateDB In Function `CallCode()`	Logical Issue	Medium	● Resolved
PRO-01	Logical Flaw In Function `filter()` Could Invoke Function From A Different Version	Logical Issue	Medium	● Resolved
SYN-01	Incorrect Generation Of `randomByte` In Function `NextSyncCommitteeIndicesFastexPhase1()`	Logical Issue, Inconsistency	Medium	● Resolved
ACT-01	Missing Nil Check Of Variable `Activity`	Volatile Code	Minor	● Resolved
ATT-01	Missing Check Of `proposerRewardDenominator` Could Possibly Lead To Division By Zero	Volatile Code	Minor	● Resolved
COR-02	Potential Overflow And Underflow	Incorrect Calculation	Minor	● Resolved

ID	Title	Category	Severity	Status
FTN-01	Potential Initialization By Frontrunner	Logical Issue	Minor	● Acknowledged
FTN-02	Missing Receive Function	Logical Issue	Minor	● Acknowledged
FTN-03	Discussion On The Mint Workflow With Function `processBurnTransaction()`	Logical Issue	Minor	● Acknowledged
MAI-01	Mainnet Could Possibly Be Misconfigured	Logical Issue	Minor	● Resolved
PRP-01	The Output Block Does Not Contain `ActivityChanges` , `TransactionsCount` , `BaseFee` , And `ExecutionHeight`	Logical Issue	Minor	● Resolved
REW-02	Possibly Incorrect Calculation Of Base Proposer Reward	Logical Issue, Inconsistency	Minor	● Resolved
33B-01	Typo In Variable Names And Function Names	Coding Style	Informational	● Resolved
3B8-01	Discussion On Value Of `SigmoidLimit`	Logical Issue	Informational	● Resolved
BEA-01	Typo In Error Messages	Coding Style	Informational	● Resolved
COB-02	Discussion On The Use Of The Sigmoid Function In Block Proposer And Sync Committee Members Selection	Logical Issue	Informational	● Resolved
COB-03	Discussion On Two Implementations Of Block Proposer And Sync Committee Selection In Different Versions	Logical Issue	Informational	● Resolved
COE-03	Inconsistency Between Implementation And Whitepaper	Logical Issue	Informational	● Resolved
DEO-02	Discussion On Contract Registration With Validators	Logical Issue	Informational	● Resolved

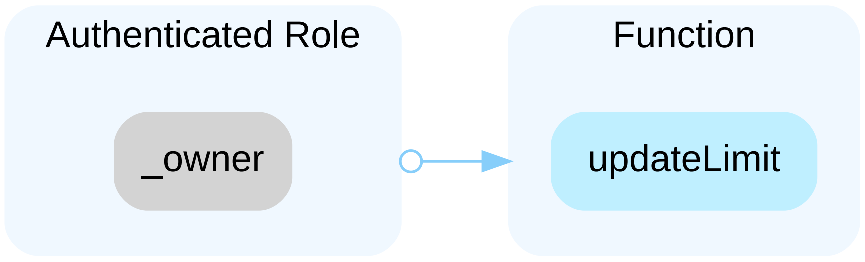
ID	Title	Category	Severity	Status
DES-02	Discussion On Inconsistency Between Deposit Contract And Its Binding	Logical Issue	Informational	● Resolved
GLOBAL-01	Current Version Does Not Contain Patch For MEV-Boost Attack	Inconsistency	Informational	● Resolved
REW-01	Discussion On The Calculation Of `BaseProposerReward`	Logical Issue	Informational	● Resolved
STF-01	Typo In The Codebase Of Execution Layer	Coding Style	Informational	● Resolved
STT-02	Typo In The Codebase Of Consensus Layer	Coding Style	Informational	● Resolved
VAL-02	Typo In Function Name `isEligibileForActivationQueue()`	Coding Style	Informational	● Resolved
VAL-03	Code Simplification In Function `RandomBytes()`	Coding Style	Informational	● Resolved
VAL-04	Inconsistency Between Implementation And Whitepaper On The Calculation Of Validator's Power	Inconsistency	Informational	● Acknowledged

FTN-04 | INITIAL DISTRIBUTION CENTRALIZATION RISK IN CONTRACT `FTNVault`

Category	Severity	Location	Status
Centralization	● **Major**	**bahamut/FTNVault.sol (bahamut): 56**	● **Acknowledged**

Description

In the contract **FTNVault**, the role **owner** has authority over the functions shown in the diagram below.



- **updateLimit(address minterAddress_, uint256 limit_)** to update the maximum amount of the native FTN token that an `minterAddress_` is able to withdraw.

According to the project design, the native FTN tokens will be initialized to the FTNVault contact in the genesis. In this case, any compromise to the **owner** account may allow a hacker to take advantage of this authority and drain the FTN tokens from the contract **FTNVault**. If the hacker controls the **owner** role, the hacker is able to call the function `updateLimit()` to set the maximum amount of the native FTN token to the hacker's account, then invokes `processBurnTransaction()` to withdraw the native FTN tokens, resulting in severe damage to the project.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Alleviation

[Fasttoken - 07/07/2023] :
The team is planning to use timelock and multisig in the future, once we have a multisig DAPP deployed on Fastex Chain, and we will share the address with you so you can verify it.

[CertiK - 07/07/2023] :
Once the timelock and multisig are applied, CertiK strongly encourages the project team to periodically revisit the private key security management.

322-01 | MISSING `Contract` WHEN PROCESSING DEPOSIT LOG

Category	Severity	Location	Status
Logical Issue	● Medium	beacon-chain/execution/log_processing.go (3226f83): 112; contracts/deposit/logs.go (3226f83): 11	● Resolved

Description

Files:

- `beacon-chain/execution/log_processing.go`
- `contracts/deposit/logs.go`

Commit:

- `3226f8330911cb8df77e775f0155b335ba771bd8`

The `ProcessDepositLog()` function in is responsible for handling the received log from the `eth1` chain and generating the deposit data object. However, there is currently no logic implemented to handle the `Contract` attribute or include it in the deposit data object.

beacon-chain/execution/log_processing.go

```
133	depositData := &ethpb.Deposit_Data{
134		Amount:                bytesutil.FromBytes8(amount),
135		PublicKey:             pubkey,
136		Signature:             signature,
137		WithdrawalCredentials: withdrawalCredentials,
138	}
```

contracts/deposit/logs.go

```go
func UnpackDepositLogData(data []byte) (pubkey, withdrawalCredentials, amount, signature, index []byte, err error) {
    reader := bytes.NewReader([]byte(DepositContractABI))
    contractAbi, err := abi.JSON(reader)
    if err != nil {
        return nil, nil, nil, nil, nil, errors.Wrap(err, "unable to generate contract abi")
    }

    unpackedLogs, err := contractAbi.Unpack("DepositEvent", data)
    if err != nil {
        return nil, nil, nil, nil, nil, errors.Wrap(err, "unable to unpack logs")
    }

    return unpackedLogs[0].([]byte), unpackedLogs[1].([]byte), unpackedLogs[2].([]byte), unpackedLogs[3].([]byte), unpackedLogs[4].([]byte), nil
}
```

Recommendation

Recommend implementing the necessary logic for handling the contract attribute and ensuring the data integrity of the deposit data.

Alleviation

[Fasttoken - 06/06/2023] :

The team resolved the finding by adding the field `Contract` in the commit `cffbd04e743737989e44cf0ebae70fd353c5a539` .

DEP-02 | POTENTIALLY OVERRIDE THE CURRENT OWNER OF CONTRACT

Category	Severity	Location	Status
Logical Issue	● Medium	beacon-chain/core/blocks/deposit.go (3226f83): 192~204	● Resolved

Description

Files:

- `beacon-chain/core/block/deposit.go`

Commit:

- `3226f8330911cb8df77e775f0155b335ba771bd8`

If a contract is associated with a previous validator, and a new validator registers the same contract, then the previous owner's contract will be set to a zero address. This creates an issue that any validator can occupy the contract with high activities, thereby gaining more power and increasing their chances of being selected as the block proposer.

```
192       if contractExist {
193           // Set zero-contract to the old owner of the contract
194           // if the contract is already presented in beacon state.
195           owner, err := beaconState.ValidatorAtIndex(contractOwner)
196           if err != nil {
197               return nil, newValidator, err
198           }
199           newVal := ethpb.CopyValidator(owner)
200           newVal.Contract = params.BeaconConfig().ZeroContract[:]
201           if err := beaconState.UpdateValidatorAtIndex(contractOwner, newVal);
 err != nil {
202               return nil, newValidator, err
203           }
204       }
```

Moreover, in the new design, a validator owns at most one contract, and the current owner of a contract cannot update the contract because once the validator has been set, then it will not be able to enter the branch to update the contract.

```
173       contractOwner, contractExist := beaconState.ValidatorIndexByContract(
bytesutil.ToBytes20(contract))
174       index, ok := beaconState.ValidatorIndexByPubkey(bytesutil.ToBytes48(pubKey)
)
175       if !ok {
176   ...
```

The auditing team would like to confirm with the Fasttoken team if the existing logic is in accordance with the design.

Recommendation

We recommend reviewing the logic again and ensuring it is as intended.

Alleviation

[Fasttoken - 06/08/2023] :

The team resolved the finding by utilizing the following logic:

- if the validator is not new, then its contract will be updated with the passed contract;
- if the validator is new and if the passed contract has been owned by a validator that has not exited, a zero contract is set to the new validator.
- if the validator is new, then if the passed contract has not been owned or the passed contract has been owned by a validator that has exited, the contract is set to the new validator.

```
        if contractExist {
            owner, err := beaconState.ValidatorAtIndexReadOnly(contractOwner)
            if err != nil {
                return nil, newValidator, err
            }
            if owner.ExitEpoch() >= epoch {
                contract = params.BeaconConfig().ZeroContract[:]
            }
        }
        if err := beaconState.AppendValidator(&ethpb.Validator{
            PublicKey:                  pubKey,
            WithdrawalCredentials:      deposit.Data.WithdrawalCredentials,
            Contract:                   contract,
            ActivationEligibilityEpoch: params.BeaconConfig().FarFutureEpoch,
            ActivationEpoch:            params.BeaconConfig().FarFutureEpoch,
            ExitEpoch:                  params.BeaconConfig().FarFutureEpoch,
            WithdrawableEpoch:          params.BeaconConfig().FarFutureEpoch,
            EffectiveBalance:           effectiveBalance,
        }); err != nil {
            return nil, newValidator, err
        }
```

The change is reflected in the commit `cffbd04e743737989e44cf0ebae70fd353c5a539` .

EVM-01 | MISSING MEMORY GAS USAGE IN ACTIVITY WHEN ADDING IT TO STATEDB IN FUNCTION `CallCode()`

Category	Severity	Location	Status
Logical Issue	● Medium	core/vm/evm.go (execution): 353	● Resolved

Description

Files:

- `core/vm/evm.go`

Commit:

- `af75d5f6c6ab5a33f6a1ac86c5c443e7be943cf1`

In the execution layer, the invocation of the function `CallCode()` changes the address's activity based on gas usage, one of which is the memory gas usage:

CallCode()

```
341         memGas, err := evm.memoryGas(input)
342         if err != nil {
343             return nil, gas, err
344         }
345         if caller.Address() != evm.Origin {
346             memGas = 0
347         }
348
349         evm.StateDB.AddActivity(addrCopy, initialGas-contract.Gas-contract.
OthersGas+memGas)
350         evm.StateDB.AddActivities(&types.Activity{
351             Address:      addrCopy,
352             Activity:     evm.StateDB.GetActivity(addrCopy),
353             DeltaActivity: initialGas - contract.Gas - contract.OthersGas,
354         })
```

However, an inconsistency occurs when adding the activity to the `evm.StateDB`. In line 349, the added activity is calculated as `initialGas-contract.Gas-contract.OthersGas+memGas`, while the `memGas` is missing in line 353 in the call of `evm.StateDB.AddActivities()`, which only accepts `initialGas - contract.Gas - contract.OthersGas` as an input.

Recommendation

We recommend adding the `MemGas` to the `DeltaActivity` of a new activity.

Alleviation

[Fasttoken - 05/11/2023] :

The team resolved the finding by removing the calculation of `memGas` and `evm.StateDB.AddActivity()` from the function `CallCode()`. The change is reflected in the commit `1b44e499f1275b821dff5f14169f4cfcd2225d22`.

PRO-01 | LOGICAL FLAW IN FUNCTION `filter()` COULD INVOKE FUNCTION FROM A DIFFERENT VERSION

Category	Severity	Location	Status
Logical Issue	● Medium	beacon-chain/rpc/prysm/v1alpha1/validator/proposer_attestations.go (33b75d4): 91~108	● Resolved

Description

Files:

- `beacon-chain/rpc/prysm/v1alpha1/validator/proposer_attestations.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

The function `filter()` is intended to filter the attestation list into valid and invalid attestations separately, which has different implementations according to different versions.

However, there is a logical flaw introduced in lines 91-108 due to the fact that `version.Altair < version.FastexPhase1` (i.e., 1 < 3):

```
 91      } else if st.Version() >= version.Altair {
 92
// Use a wrapper here, as go needs strong typing for the function signature.
 93          attestationProcessor = func(ctx context.Context, st state.BeaconState,
 attestation *ethpb.Attestation) (state.BeaconState, error) {
 94              totalBalance, err := helpers.TotalActiveBalance(st)
 95              if err != nil {
 96                  return nil, err
 97              }
 98              return altair.ProcessAttestationNoVerifySignature(ctx, st,
 attestation, totalBalance)
 99          }
100      } else if st.Version() >= version.FastexPhase1 {
101
// Use a wrapper here, as go needs strong typing for the function signature.
102          attestationProcessor = func(ctx context.Context, st state.BeaconState,
 attestation *ethpb.Attestation) (state.BeaconState, error) {
103              totalBalance, err := helpers.TotalActiveBalance(st)
104              if err != nil {
105                  return nil, err
106              }
107              return fastexphase1.ProcessAttestationNoVerifySignature(ctx, st,
 attestation, totalBalance)
108          }
```

The branch `st.Version() >= version.FastexPhase1` is unreachable because any version not less than `version.Altair` will enter the branch `st.Version() >= version.Altair` in line 91. In this case, if the current version is in the post-FastexPhase1, it will use the function `altair.ProcessAttestationNoVerifySignature()` instead of the function `fastexphase1.ProcessAttestationNoVerifySignature()` as the `attestationProcessor`, which could lead to an unexpected result. For example, different implementations of the function `RewardProposer()` will be invoked.

Recommendation

Recommend reconstructing the logic so that the function `fastexphase1.ProcessAttestationNoVerifySignature()` will be used in the post-FastexPhase1.

Alleviation

[Fasttoken - 05/25/2023] :

The team heeded the advice and resolved the finding by removing the branch `st.Version() >= version.FastexPhase1` so that all the versions satisfying the condition `st.Version() >= version.Altair` will enter the same branch using the same implementation. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8`.

SYN-01 INCORRECT GENERATION OF `randomByte` IN FUNCTION `NextSyncCommitteeIndicesFastexPhase1()`

Category	Severity	Location	Status
Logical Issue, Inconsistency	● Medium	beacon-chain/core/altair/sync_committee.go (33b75d4): 127~130, 200~201, 209	● Resolved

Description

Files:

- `beacon-chain/core/altair/sync_committee.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

The incorrect generation of `randByte` allows any active validator to be selected in the sync committee regardless of their effective balances, which does not align with the consensus algorithm.

The function `NextSyncCommitte()` is used to select the sync committee members from the active validators. In the current code, Fasttoken implements two algorithms according to the version.

```go
func NextSyncCommittee(ctx context.Context, s state.BeaconState) (*ethpb.
SyncCommittee, error) {
    var indices []primitives.ValidatorIndex
    var err error
    if s.Version() < version.FastexPhase1 {
        indices, err = NextSyncCommitteeIndices(ctx, s)
    } else {
        indices, err = NextSyncCommitteeIndicesFastexPhase1(ctx, s)
    }
...
```

If the version is less than the `FastexPhase1`, it uses the custom algorithm that applies the validator power by invoking the function `NextSyncCommitteeIndices()`. On the other hand, if the version is in post `FastexPhase1`, the algorithm inherits the original one from Ethereum Proof of Stake, which is implemented in the function `NextSyncCommitteeIndicesFastexPhase1()`.

Both algorithms use the same randomness generation and the same `maxRandomByte` (= 65535). In the function `NextSyncCommitteeIndices()`, the `randomBytes` is generated by two bytes, so `randomBytes` is in the range of 0 and 65535.

NextSyncCommitteeIndices()

```
        b := append(seed[:], bytesutil.Bytes8(uint64(i.Div(16)))...)
        hash := hashFunc(b)
        bytes2 := append([]byte{}, hash[i%16], hash[16+i%16])
        randomBytes := new(big.Float).SetUint64(uint64(bytesutil.FromBytes2(
bytes2)))
```

However, the `randomByte` in the function `NextSyncCommitteeIndicesFastexPhase1()` only has one byte, which is in the range of 0 and 255. In this case, the ratio `randomByte` / `maxRandomByte` is too small which allows almost all validators to be selected regardless of their effective balances. In other words, the effective balance does not affect the chance of a validator to be selected.

NextSyncCommitteeIndicesFastexPhase1()

```
        b := append(seed[:], bytesutil.Bytes8(uint64(i.Div(32)))...)
        randomByte := hashFunc(b)[i%32]
        cIndex := indices[sIndex]
        v, err := s.ValidatorAtIndexReadOnly(cIndex)
        if err != nil {
            return nil, err
        }

        effectiveBal := v.EffectiveBalance()
        if effectiveBal*maxRandomByte >= cfg.MaxEffectiveBalance*uint64(
randomByte) {
            cIndices = append(cIndices, cIndex)
        }
```

Proof of Concept

To demonstrate the scenario, the auditing team uses the following test script:

1. Initialize 512 validators with `minDepositAmount` == `1e9 / 8`;
2. Normally, a validator needs a `16e9` deposit amount to be active and the max effective balance is `32e9` in PoS. The number `1e9 / 8` is used here to indicate the effective balance check can be bypassed with a very small effective balance;
3. Invoke the function `NextSyncCommitteeIndicesFastexPhase1()` for the testing.

Test Script:

```
package altair_test

import (
    "context"
    "fmt"
    "testing"
    "time"

    "github.com/prysmaticlabs/prysm/v3/beacon-chain/core/altair"
    "github.com/prysmaticlabs/prysm/v3/beacon-chain/core/helpers"
    "github.com/prysmaticlabs/prysm/v3/beacon-chain/state"
    state_native "github.com/prysmaticlabs/prysm/v3/beacon-chain/state/state-native"
    "github.com/prysmaticlabs/prysm/v3/config/params"
    "github.com/prysmaticlabs/prysm/v3/consensus-types/primitives"
    "github.com/prysmaticlabs/prysm/v3/crypto/bls"
    ethpb "github.com/prysmaticlabs/prysm/v3/proto/prysm/v1alpha1"
    "github.com/prysmaticlabs/prysm/v3/testing/assert"
    "github.com/prysmaticlabs/prysm/v3/testing/require"
    prysmTime "github.com/prysmaticlabs/prysm/v3/time"
)

func TestNextSyncCommitteeIndicesFastexPhase1(t *testing.T) {
    getState := func(t *testing.T, count uint64) state.BeaconState {
        validators := make([]*ethpb.Validator, count)
        for i := 0; i < len(validators); i++ {
            validators[i] = &ethpb.Validator{
                ExitEpoch:        params.BeaconConfig().FarFutureEpoch,
                EffectiveBalance: params.BeaconConfig().MinDepositAmount / 8,
            }
        }
        st, err := state_native.InitializeFromProtoAltair(&ethpb.BeaconStateAltair{
            Validators:  validators,
            RandaoMixes: make([][]byte,
params.BeaconConfig().EpochsPerHistoricalVector),
        })
        require.NoError(t, err)
        return st
    }
    st := getState(t, 512)
    got, err := altair.NextSyncCommitteeIndicesFastexPhase1(context.Background(),
st)
    require.NoError(t, err)
    fmt.Printf("Number of Sync commeetiee members is: %d out of %d members\n",
len(got), 512)
}
```

Result:

```
=== RUN   TestNextSyncCommitteeIndicesFastexPhase1
Number of Sync commeetiee members is: 512 out of 512 members
--- PASS: TestNextSyncCommitteeIndicesFastexPhase1 (0.76s)
PASS
```

The result shows all 512 validators have been selected even though their effective balances are very small.

Recommendation

Recommend changing the generation of `randByte` in the function `NextSyncCommitteeIndicesFastexPhase1()` to have two bytes in order to align with the `maxRandomByte` .

Alleviation

[Fasttoken - 05/25/2023] :

The team heeded the advice and resolved the finding by removing the implementation when `s.Version() < version.FastexPhase1` and changing the `maxRandomByte` from `uint64(1<<16 - 1)` to `uint64(1<<8 - 1)` :

```
23  const maxRandomByte = uint64(1<<8 - 1)
```

The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

ACT-01 | MISSING NIL CHECK OF VARIABLE `Activity`

Category	Severity	Location	Status
Volatile Code	● Minor	beacon-chain/core/blocks/activities.go (consensus): 42	● Resolved

Description

Files:

- `beacon-chain/core/blocks/activities.go`

Commit:

- `3b8da2895d7067405b54c0829eee7e044a0f978e`

The function `ProcessBlockActivities()` is intended to process the activities in a block, which invokes the function `ProcessActivity()` with each `activity` from the block. Since the activities are fetched from the execution layer, they could possibly contain some nil value. If the `activity` is nil, then fetching `ContractAddress` from the `activity` in line 42 of the function `ProcessActivity()` will lead to a runtime panic.

```go
34  func ProcessActivity(
35      ctx context.Context,
36      beaconState state.BeaconState,
37      activity *ethpb.ActivityChange,
38  ) (state.BeaconState, error) {
39      ctx, span := trace.StartSpan(ctx, "core.ProcessActivtiyNoVerifySignature")
40      defer span.End()
41
42      contract := bytesutil.ToBytes20(activity.ContractAddress)
43      idx, ok := beaconState.ValidatorIndexByContractAddress(contract)
44      if !ok {
45          nonStakersGas := beaconState.NonStakersGasPerEpoch()
46          if err := beaconState.SetNonStakersGasPerEpoch(nonStakersGas + activity
.DeltaActivity); err != nil {
47              return nil, err
48          }
49          return beaconState, nil
50      }
51  ...
```

Recommendation

Recommend adding the nil check of the activity to ensure no nil value is passed into the function `ProcessActivity()`.

Alleviation

[Fasttoken - 05/25/2023] :

The team heeded the advice and resolved the finding by adding the nil check of the activity. Additionally the file has been renamed from `activities.go` to `activity_changes.go` :

beacon-chain/core/blocks/activity_changes.go

```go
func ProcessActivityChanges(
    ctx context.Context,
    beaconState state.BeaconState,
    activityChanges []*ethpb.ActivityChange,
) (state.BeaconState, error) {
    var err error
    for _, ac := range activityChanges {
        if ac == nil || ac.ContractAddress == nil {
            return nil, errors.New("got a nil activity change in block")
        }
        beaconState, err = ProcessActivityChange(ctx, beaconState, ac)
        if err != nil {
            return nil, errors.Wrapf(err,
"could not process activity change from 0x%x", ac.ContractAddress)
        }
    }
    return beaconState, nil
}
```

The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

ATT-01 | MISSING CHECK OF `proposerRewardDenominator` COULD POSSIBLY LEAD TO DIVISION BY ZERO

Category	Severity	Location	Status
Volatile Code	● Minor	beacon-chain/core/fastex-phase1/attestation.go (33b75d4): 205, 213	● Resolved

Description

Files:

- `beacon-chain/core/fastex-phase1/attestation.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

The function `RewardProposer()` is intended to calculate the reward for the block proposer, which accepts the parameters, `proposerRewardNumerator` and `proposerRewardDenominator` from the return values of function `EpochParticipation()`. If the passed `indices` is empty in the loop of the `EpochParticipation()`, both `proposerRewardNumerator` and `proposerRewardDenominator` will be 0 and the returned error is nil.

```go
func EpochParticipation(
    beaconState state.BeaconState,
    indices []uint64,
    epochParticipation []byte,
    participatedFlags map[uint8]bool,
    totalBalance uint64,
) (uint64, uint64, []byte, error) {
    cfg := params.BeaconConfig()
    sourceFlagIndex := cfg.TimelySourceFlagIndex
    targetFlagIndex := cfg.TimelyTargetFlagIndex
    headFlagIndex := cfg.TimelyHeadFlagIndex
    proposerRewardNumerator := uint64(0)
    proposerRewardDenominator := uint64(0)
    for _, index := range indices {
...
    }
    return proposerRewardNumerator, proposerRewardDenominator, epochParticipation, nil
}
```

In this case, error handling in lines 99-101 and 109-110 of the function `SetParticipationAndRewardProposer()` will be bypassed.

```go
 84  func SetParticipationAndRewardProposer(
 85      ctx context.Context,
 86      beaconState state.BeaconState,
 87      targetEpoch primitives.Epoch,
 88      indices []uint64,
 89      participatedFlags map[uint8]bool,
 90      totalBalance uint64,
 91  ) (state.BeaconState, error) {
 92      var proposerRewardNumerator uint64
 93      var proposerRewardDenominator uint64
 94      currentEpoch := time.CurrentEpoch(beaconState)
 95      var stateErr error
 96      if targetEpoch == currentEpoch {
 97          stateErr = beaconState.ModifyCurrentParticipationBits(func(val []byte) ([]byte, error) {
 98              propRewardNum, propRewardDenom, epochParticipation, err := EpochParticipation(beaconState, indices, val, participatedFlags, totalBalance)
 99              if err != nil {
100                  return nil, err
101              }
102              proposerRewardNumerator = propRewardNum
103              proposerRewardDenominator = propRewardDenom
104              return epochParticipation, nil
105          })
106      } else {
107          stateErr = beaconState.ModifyPreviousParticipationBits(func(val []byte) ([]byte, error) {
108              propRewardNum, propRewardDenom, epochParticipation, err := EpochParticipation(beaconState, indices, val, participatedFlags, totalBalance)
109              if err != nil {
110                  return nil, err
111              }
112              proposerRewardNumerator = propRewardNum
113              proposerRewardDenominator = propRewardDenom
114              return epochParticipation, nil
115          })
116      }
117      if stateErr != nil {
118          return nil, stateErr
119      }
120
121      if err := RewardProposer(ctx, beaconState, proposerRewardNumerator, proposerRewardDenominator); err != nil {
122          return nil, err
123      }
124
125      return beaconState, nil
126  }
```

In addition, the error handling in lines 117-119 will also be bypassed, allowing the function `RewardProposer()` with `proposerRewardDenominator` as 0 to be invoked in the function `SetParticipationAndRewardProposer()`. Therefore, the parameter `proposerRewardDenominator` passed in the `RewardProposer()` is 0.

Recommendation

To avoid the potential corner case that causes division-by-zero runtime panic, recommend adding an extra check in the function `RewardProposer()` to ensure the passed `proposerRewardDenominator` is nonzero.

Alleviation

[Fasttoken - 05/25/2023] :

The team heeded the advice and resolved the finding by adding the check of `proposerRewardDenominator` in the function `RewardProposer()` , which has been incorporated in the file `beacon-chain/core/altair/attestation.go` as the folder `beacon-chain/core/fastex-phase1` has been removed:

beacon-chain/core/altair/attestation.go

```go
func RewardProposer(ctx context.Context, beaconState state.BeaconState, proposerRewardNumerator, proposerRewardDenominator uint64) error {
    cfg := params.BeaconConfig()
    totalPower, totalEffectivePower, err := helpers.Powers(ctx, beaconState)
    if err != nil {
        return err
    }
    baseProposerReward, err := BaseProposerReward(beaconState, totalPower, totalEffectivePower)
    if err != nil {
        return err
    }

    proposerReward := baseProposerReward * (cfg.WeightDenominator - cfg.SyncRewardWeight) / cfg.WeightDenominator
    if proposerRewardDenominator == 0 {
        proposerReward = 0
    } else {
        proposerReward = proposerReward * proposerRewardNumerator / proposerRewardDenominator
    }

    i, err := helpers.BeaconProposerIndex(ctx, beaconState)
    if err != nil {
        return err
    }

    return helpers.IncreaseBalance(beaconState, i, proposerReward)
}
```

The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

COR-02 | POTENTIAL OVERFLOW AND UNDERFLOW

Category	Severity	Location	Status
Incorrect Calculation	● Minor	core/state/state_object.go (execution): 400, 412; core/state/state_transition.go (execution): 406, 415~417, 423~425, 423~425, 427	● Resolved

Description

Files:

- `core/state/state_object.go`
- `core/state/state_transition.go`

Commit:

- `af75d5f6c6ab5a33f6a1ac86c5c443e7be943cf1`

There are no overflow and underflow protections in the following functions, making it possible for overflow/underflow to occur and could possibly lead to inaccurate calculations.

core/state/state_object.go

```
400  func (s *stateObject) AddActivity(amount uint64) {
401      if amount == 0 {
402          if s.empty() {
403              s.touch()
404          }
405          return
406      }
407
408      s.SetActivity(s.Activity() + amount)
409  }
410
411  // SubActivity remove some amount of activity to s's activity
412  func (s *stateObject) SubActivity(amount uint64) {
413      if amount == 0 {
414          if s.empty() {
415              s.touch()
416          }
417          return
418      }
419
420      s.SetActivity(s.Activity() - amount)
421  }
```

core/state_transition.go

```go
func (st *StateTransition) refundActivity(refund uint64) {
    if refund == 0 {
        return
    }

    totalRefund := refund
    totalActivityByContract := make(map[common.Address]uint64)
    totalRefundsByContracts := make(map[common.Address]uint64)
    currentActivities := st.state.GetCurrentActivities()
    for _, act := range currentActivities {
        totalActivityByContract[act.Address] += act.DeltaActivity
    }
    var proportion []float64
    for _, act := range currentActivities {
        proportion = append(proportion, float64(act.DeltaActivity)/float64(totalActivityByContract[act.Address]))
    }

    for i, act := range currentActivities {
        if i == len(currentActivities)-1 {
            totalRefundsByContracts[act.Address] += refund
            act.DeltaActivity -= refund
            act.Activity -= totalRefundsByContracts[act.Address]
            st.state.SubActivity(act.Address, refund)
            log.Debug("Refunded contract activity", "activity", refund, "addr", act.Address)
        } else {
            totalRefundByContract := float64(totalRefund*st.state.GetRefundsByContract(act.Address)) / float64(st.state.GetRefund())
            refundAct := uint64(totalRefundByContract * proportion[i])
            totalRefundsByContracts[act.Address] += refundAct
            act.DeltaActivity -= refundAct
            act.Activity -= totalRefundsByContracts[act.Address]
            st.state.SubActivity(act.Address, refundAct)
            refund -= refundAct
            log.Debug("Refunded contract activity", "activity", totalRefundByContract, "addr", act.Address)
        }
    }
}
```

Recommendation

We recommend adding overflow and underflow protections for these functions. Additionally, we also recommend reviewing all other functions and ensuring overflow and underflow protections are applied.

Alleviation

[Fasttoken - 05/11/2023] :

The team resolved the finding by removing the functions `AddActivity()` and `SubActivity()` from the codebase. The change is reflected in the commit `1b44e499f1275b821dff5f14169f4cfcd2225d22` .

[CertiK - 05/11/2023] :

The function `refundActivity()` in the file **core/state_transition.go** has been modified in the commit `1b44e499f1275b821dff5f14169f4cfcd2225d22` , but the recommendation is still able to be applied.

[Fasttoken - 07/06/2023] :

The team resolved the issue at the function `refundActivity()` of the file **core/state_transition.go**. The change is reflected in the commit `3d669ac92faa0747a2aa2e8905e46d39c563d114` .

FTN-01 | POTENTIAL INITIALIZATION BY FRONTRUNNER

Category	Severity	Location	Status
Logical Issue	● Minor	bahamut/FTNVault.sol (bahamut): 43	● Acknowledged

Description

Files:

- `bahamut/FTNVault.sol`

Commit:

- `1f2392be6927c2227a0061a5c7c9f7c937545971`

In the contract `FTNVault`, the function `initialize()` can be called by anyone due to no access restriction, which enables anyone to initialize the contract, and gain ownership of the contract. Malicious users could observe the pending transaction which will execute the `initialize()` function in the mempool, and launch a similar transaction to front-run the pending transaction.

```
    function initialize(bytes32 burnTxHash_) public {

        require(!initialized, 'Contract has already been initialized');
        initialized = true;

        _transferOwnership(msg.sender);
        burnTransactionHashes[burnTxHash_] = true;

        uint256 amount = 1000 * 10**18;
        emit BurnTransactionProcessed(burnTxHash_, msg.sender, amount);
    }
```

In the case that the contract has some native FTN tokens after the deployment, then the malicious users that control the contract will be able to drain the contract via the functions `updateLimit()` and `processBurnTransaction()`.

Recommendation

Consider the following modification to the function `initialize()`:

- add access control to the function `initialize()` so that only the deployer is able to call it;
- set a new parameter to accept the new owner and pass the new owner to the function `_transferOwnership()`.

Alleviation

[Fasttoken - 05/04/2023] :

The team acknowledged the finding. This is impossible simply due to the fact that only one account/address (which the team has) has access to native FTNs to do the mentioned transaction. There is literally no other FTNs available to any potential malicious users, even if they frontrun it, they cannot execute the transaction without native FTNs.

FTN-02 | MISSING RECEIVE FUNCTION

Category	Severity	Location	Status
Logical Issue	● Minor	bahamut/FTNVault.sol (bahamut): 19	● Acknowledged

Description

Files:

- `bahamut/FTNVault.sol`

Commit:

- `1f2392be6927c2227a0061a5c7c9f7c937545971`

The contract `FTNVault.sol` serves as a vault of native FTN tokens to redeem the same amount of FTN tokens that the user has burnt on Ethereum.

However, no receive, fallback, or any payable function is implemented in the contract to accept the native FTN tokens. In this case, there is no FTN token in the vault except for tokens obtained from the self-destruct of other contracts or before the deployment. Both methods do not align with the current design because the amount of native FTN tokens is determined by the burnt amount on Ethereum.

Proof of Concept

To demonstrate the scenario, the auditing team uses the following test script with the Foundry framework:

1. Send Alice 1000 ether;
2. Initialize the contract FTNVault;
3. Alice sends 100 ether to the contract FTNVault.

Test Script

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.15;

//import "forge-std/Script.sol";
import "forge-std/Test.sol";
import "src/FTNVault.sol";

contract PoC is Test {
    address Alice = address(1);

    function setUp() public {
        vm.deal(Alice, 1000 ether);
    }

    function testSendFTN() public {
        FTNVault vault = new FTNVault();
        emit log_string("-------------- Before Ether Sent --------------");
        emit log_named_uint("Balance of Alice ", address(Alice).balance / 1 ether);
        emit log_named_uint("Balance of FTNVault ", address(vault).balance / 1
ether);
        emit log_named_address("The vault address ", address(vault));

        // sent 100 ether from Alice to the vault
        vm.startPrank(Alice);
        payable(address(vault)).transfer(100 ether);
        vm.stopPrank();

        emit log_string("-------------- After Ether Sent --------------");
        emit log_named_uint("Balance of Alice ", address(Alice).balance / 1 ether);
        emit log_named_uint("Balance of FTNVault ", address(vault).balance / 1
ether);
        emit log_named_address("The vault address ", address(vault));
    }
}
```

Result

```
Running 1 test for test/FTNVault.t.sol:PoC
[FAIL. Reason: EvmError: Revert] testSendFTN() (gas: 481340)
Logs:
  -------------- Before Ether Sent --------------
  Balance of Alice : 1000
  Balance of FTNVault : 0
  The vault address : 0xce71065d4017f316ec606fe4422e11eb2c47c246

Traces:
  [5138] PoC::setUp()
    ├─ [0] VM::deal(0x0000000000000000000000000000000000000001, 
10000000000000000000000)
    │   └─ ← ()
    └─ ← ()

  [481340] PoC::testSendFTN()
    ├─ [428584] → new FTNVault@"0xce71…c246"
    │   ├─ emit OwnershipTransferred(previousOwner: 
0x0000000000000000000000000000000000000000, newOwner: PoC: 
[0xb4c79dab8f259c7aee6e5b2aa729821864227e84])
    │   └─ ← 2022 bytes of code
    ├─ emit log_string(: "-------------- Before Ether Sent --------------")
    ├─ emit log_named_uint(key: "Balance of Alice ", val: 1000)
    ├─ emit log_named_uint(key: "Balance of FTNVault ", val: 0)
    ├─ emit log_named_address(key: "The vault address ", val: FTNVault: 
[0xce71065d4017f316ec606fe4422e11eb2c47c246])
    ├─ [0] VM::startPrank(0x0000000000000000000000000000000000000001)
    │   └─ ← ()
    ├─ [45] FTNVault::fallback{value: 1000000000000000000000}()
    │   └─ ← "EvmError: Revert"
    └─ ← "EvmError: Revert"

Test result: FAILED. 0 passed; 1 failed; finished in 786.35µs

Failed tests:
[FAIL. Reason: EvmError: Revert] testSendFTN() (gas: 481340)

Encountered a total of 1 failing tests, 0 tests succeeded
```

The result shows that the native token transfer from Alice to the vault is reverted.

Recommendation

Recommend adding the receive function in the contract to accept the native FTN token transfer.

Alleviation

[Fasttoken - 05/09/2023] :

The team acknowledged the finding and decide not to make any change to the current version as the initial FTN tokens will be sent to the contract in the genesis.

FTN-03 | DISCUSSION ON THE MINT WORKFLOW WITH FUNCTION `processBurnTransaction()`

Category	Severity	Location	Status
Logical Issue	● Minor	bahamut/FTNVault.sol (bahamut): 64	● Acknowledged

Description

Files:

- `bahamut/FTNVault.sol`

Commit:

- `1f2392be6927c2227a0061a5c7c9f7c937545971`

The contract `FTNVault` serves as a vault of native FTN tokens to redeem the same amount of FTN tokens that the user has burnt on Ethereum. By design, the user burns the FTN token on Ethereum and redeems the same amount of the burnt FTN token from the contract `FTNVault` through the function `processBurnTransaction()` with the burn transaction:

```
    function processBurnTransaction(bytes32 burnTxHash_, address recipient_, uint256 amount_) external {

        require(initialized, 'Contract has not been initialized');
        require(amount_ <= limits[msg.sender], 'Limit exceeded');
        limits[msg.sender] -= amount_;
        _processBurnTransaction(burnTxHash_, recipient_, amount_);
    }
```

However, the current implementation seems to miss some logic to validate the burner and amount that is burnt on Ethereum.

1. There is no validation to ensure the passed amount is the amount burnt in the transaction;
2. Similarly, no validation to make sure the user is related to the burner that burns the FTN tokens. The only way is to set the limits to a user via the function `updateLimit()` in a centralized manner;
3. No validation to ensure the passed `burnTxHash_` is actually a burn transaction that happened on Ethereum; Any user that has the limit is able to withdraw all the allowed balance by passing an unused `bytes32`.

Recommendation

The auditing team would like to understand the workflow to redeem the FTN tokens from the burn transactions on Ethereum.

Alleviation

[Fasttoken - 05/09/2023] :

The team acknowledged the finding. As discussed, this issue remains as it is since there is no good way to validate the TRX from Ethereum on Fastex Chain.

MAI-01 | MAINNET COULD POSSIBLY BE MISCONFIGURED

Category	Severity	Location	Status
Logical Issue	● Minor	config/params/mainnet_config.go (consensus): 92, 93	● Resolved

Description

Files:

- `config/params/mainnet_config.go`
- `config/params/minimal_config.go`

Commit:

- `3b8da2895d7067405b54c0829eee7e044a0f978e`

The parameters `MaxEffectiveBalance` and `EjectionBalance` were properly set in the configuration file `testnet_fastex_chain_config.go` . However, they were not updated in the configuration file `mannet_config.go` to accommodate the new features and functionality. A misconfiguration could cause errors or bugs that could negatively impact the functionality of the project.

```
92    MaxEffectiveBalance:        32 * 1e9,
93    EjectionBalance:            16 * 1e9,
```

In addition, the below parameters in the configuration file `minimal_config.go` are not properly set.

```
20     MinGenesisTime:                  1606824000, // Dec 1, 2020, 12pm UTC.
```

```
25      minimalConfig.MinDepositAmount = 1e9
26      minimalConfig.MaxEffectiveBalance = 32e9
27      minimalConfig.EjectionBalance = 16e9
28      minimalConfig.EffectiveBalanceIncrement = 1e9
```

Recommendation

We recommend reviewing the configuration files `mannet_config.go` and `minimal_config.go` to ensure that all relevant configuration parameters are properly set.

Alleviation

[Fasttoken - 05/25/2023] :

The team resolved the finding by changing the balance related constants in file `mainnet_config.go` :

```
    // Gwei value constants.
    MinDepositAmount:          1 * 1e9,
    MaxEffectiveBalance:       8192 * 1e9,
    EjectionBalance:           4096 * 1e9,
    EffectiveBalanceIncrement: 1 * 1e9,
```

The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

[CertiK - 05/25/2023] :

The constants in `minimal_config.go` has not been modified accordingly.

[Fasttoken - 07/06/2023] :

The team resolved the finding by making the changes in the commit `8198a02d28dee2b7485610279bcf24e4f0a2bf54` .

PRP-01 | THE OUTPUT BLOCK DOES NOT CONTAIN `ActivityChanges` , `TransactionsCount` , `BaseFee` , AND `ExecutionHeight`

Category	Severity	Location	Status
Logical Issue	● Minor	beacon-chain/rpc/prysm/v1alpha1/validator/proposer_bellatrix.go (3226f83): 306~317	● Resolved

Description

Files:

- `beacon-chain/rpc/prysm/v1alpha1/validator/proposer_bellatrix.go`

Commit:

- `3226f8330911cb8df77e775f0155b335ba771bd8`

The function `unblindBuilderBlock()` retrieves the full payload block using the input blind block. However, the output block does not contain the fields `ActivityChanges` , `TransactionsCount` , `BaseFee` , And `ExecutionHeight` .

```
        Body: &ethpb.BeaconBlockBodyBellatrix{
            RandaoReveal:      psb.Block.Body.RandaoReveal,
            Eth1Data:          psb.Block.Body.Eth1Data,
            Graffiti:          psb.Block.Body.Graffiti,
            ProposerSlashings: psb.Block.Body.ProposerSlashings,
            AttesterSlashings: psb.Block.Body.AttesterSlashings,
            Attestations:      psb.Block.Body.Attestations,
            Deposits:          psb.Block.Body.Deposits,
            VoluntaryExits:    psb.Block.Body.VoluntaryExits,
            SyncAggregate:     agg,
            ExecutionPayload:  pbPayload,
        },
```

Recommendation

Recommend reviewing the logic again and ensuring all fields are included in the output block.

Alleviation

[Fasttoken - 06/09/2023] :

The team resolved the finding by adding the missing fields in the commit `88551682018d09cf69ab604d8ccb42e7024564eb`.

REW-02 | POSSIBLY INCORRECT CALCULATION OF BASE PROPOSER REWARD

Category	Severity	Location	Status
Logical Issue, Inconsistency	● Minor	beacon-chain/core/altair/reward.go (3226f83): 65	● Resolved

Description

Files:

- `beacon-chain/core/altair/reward.go`

Commit:

- `3226f8330911cb8df77e775f0155b335ba771bd8`
- `8198a02d28dee2b7485610279bcf24e4f0a2bf54`

The fasttoken introduces a novel proposer base reward calculation based on the validator's power via the activity score associated with the validator's registered contracts. The function `BaseProposerReward()` is used to compute the base proposer reward defined in the whitepaper as follows:

$BPR = \frac{(A+T)*bf}{W*n*gwei}$,

where $A = \sum_{i=1}^{n} ea_i$ is the total effective activities of the n validators and T is the transaction constant gas in the window of 1575 epochs. The bf is the base fee of the block and W is the epoch window size 1575, n is the number of validators and $gwei$ is the constant 10^9.

Therefore, the base proposer reward in each epoch is $BPR * s$, where s (= 32) is the number of slots in an epoch.

On the other hand, the total validator base reward per epoch is given by:

$BR_{total} = \frac{B*f}{\sqrt{B}} = f * \sqrt{B}$, where B is the total active balance, and f is the constant 156 (according to the `config/params/mainnet_config.go` in the commit `8198a02d28dee2b7485610279bcf24e4f0a2bf54`).

Assume that the current number of validators is the target number 4096, and each of them has an effective balance of $8192 * 10^9$. Then the base reward is

$156 * \sqrt{4096 * 8192 * 10^9} = 0.9 * 10^9 gwei.$

According to the design, this reward will be distributed to the validators for attestation rewards and participating sync committees. In the Ethereum PoS, $1/7$ of the reward (i.e., $0.13 * 10^9$) is granted to the block proposers for proposing blocks.

The fasttoken attempts to use the base proposer reward in each epoch (i.e., $32 * BPR$) to replace the $1/7$ of the total validator base reward per epoch as the reward to the block proposers. In this case, assume that each block has a half load

(15M gas consumed) and the base fee is $100 * gwei$, then the base proposer reward in each epoch is

$$32 * \frac{(A+T)*bf}{W*n*gwei} = 32 * \frac{(32*15M)*100*gwei}{4096*gwei} = 0.395 * 10^9 gwei,$$

Therefore, the calculated reward is close to the value in the new design.

However, in the implementation of the function `BaseProposerReward()` :

```go
func BaseProposerReward(s state.ReadOnlyBeaconState, totalPower,
totalEffectivePower uint64) (uint64, error) {
    activity, err := helpers.TotalEffectiveActivity(s)
    if err != nil {
        return 0, errors.Wrap(err,
"could not calculate total effective activity")
    }

    sharedActivity := s.SharedActivity()
    if sharedActivity == nil {
        return 0, errors.New("nil shared activity in state")
    }

    period := uint64(params.BeaconConfig().EpochsPerActivityPeriod)
    slotsPerEpoch := uint64(params.BeaconConfig().SlotsPerEpoch)
    denominator := period * period * slotsPerEpoch * slotsPerEpoch
    transactionsGas := sharedActivity.TransactionsGasPerPeriod
    baseFee := sharedActivity.BaseFeePerPeriod
    reward := baseFee * (activity + transactionsGas) / denominator
    if totalPower == 0 {
        return reward, nil
    }

    return reward * totalEffectivePower / totalPower, nil
}
```

The reward does not align with the formula in the whitepaper as the reward is not divided by the number of active validators. Actually, the formula derived from the above code on average is as follows:

$$(A+T) * bf = 15M * 100 gwei = 1.5 * 10^9 gwei.$$

Recommendation

Recommend revisiting the calculation of the base proposer reward and implementing the correct formula in the whitepaper if it is the intended design.

Alleviation

[Fasttoken - 07/10/2023] :

The team provided additional design documentation to confirm this is the intended design that the base proposer reward is the average burned amount of FTNs tokens in a single block during period. The whitepaper will be updated accordingly soon.

33B-01 | TYPO IN VARIABLE NAMES AND FUNCTION NAMES

Category	Severity	Location	Status
Coding Style	● Informational	beacon-chain/execution/activities_processing.go (33b75d4): 62; beacon-chain/rpc/apimiddleware/structs.go (33b75d4): 1019; beacon-chain/rpc/prysm/v1alpha1/validator/proposer_eth1data.go (33b75d4): 119; config/features/config.go (33b75d4): 73, 209, 210, 211; config/features/flags.go (33b75d4): 85, 176; config/params/config.go (33b75d4): 145; config/params/mainnet_config.go (33b75d4): 202; validator/keymanager/remote-web3signer/v1/custom_mappers.go (33b75d4): 288; validator/keymanager/remote-web3signer/v1/web3signer_types.go (33b75d4): 183	● Resolved

Description

Files:

- `config/params/config.go`
- `config/params/mainnet_config.go`
- `config/features/config.go`
- `config/features/flags.go`
- `beacon-chain/rpc/prysm/v1alpha1/validator/proposer_eth1data.go`
- `validator/keymanager/remote-web3signer/v1/custom_mappers.go`
- `validator/keymanager/remote-web3signer/v1/web3signer_types.go`
- `beacon-chain/execution/activities_processing.go`
- `beacon-chain/rpc/apimiddleware/structs.go`
- `beacon-chain/node/node.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

Variable Names

According to the context, the variable `EpochsPerAcrivityUpdate` should be **`EpochsPerActivityUpdate`** in the following two places:

1. line 145 in the file `config/params/config.go` ;
2. line 202 in the file `config/params/mainnet_config.go` .

The variable `DisableStakinContractCheck` (`disableStakinContractCheck`) should be **`DisableStakingContractCheck`** (**`disableStakingContractCheck`**) in the following places:

1. line 73, 209, 210, and 211 in the file `config/features/config.go` ;
2. line 85 and 176 in the file `config/features/flags.go` ;
3. line 119 in the file `beacon-chain/rpc/prysm/v1alpha1/validator/proposer_eth1data.go` .

The variable `ContractAddres` should be **`ContractAddress`** in the following places:

1. line 288 in the file `validator/keymanager/remote-web3signer/v1/custom_mappers.go` ;
2. line 183 in the file `validator/keymanager/remote-web3signer/v1/web3signer_types.go` .

The variable `activiyChanges` should be **`activityChanges`** in the following place:

- line 62 in the file `beacon-chain/execution/activities_processing.go` .

The variable `EffectivtActivity` should be **`EffectiveActivity`** in the following place:

- line 1019 in the file `beacon-chain/rpc/apimiddleware/structs.go` .

Function Names

The function name `registerDeterminsticGenesisService()` should be **registerDeterministicGenesisService()** in the following places:

- line 230 and 920 in the file `beacon-chain/node/node.go` .

Recommendation

Recommend correcting the aforementioned typos to improve the code readability.

Alleviation

[Fasttoken - 05/25/2023] :
The team heeded the advice and resolved the finding by either removing the relevant code or correcting the typo. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

[CertiK - 05/25/2023] :
The following typos have not been corrected：

The variable `DisableStakinContractCheck` (`disableStakinContractCheck`) should be **`DisableStakingContractCheck`** (**`disableStakingContractCheck`**) in the following places:

1. line 73, 209, 210, and 211 in the file `config/features/config.go` ;
2. line 85 and 176 in the file `config/features/flags.go` ;
3. line 119 in the file `beacon-chain/rpc/prysm/v1alpha1/validator/proposer_eth1data.go` .

The function name `registerDeterminsticGenesisService()` should be **registerDeterministicGenesisService()** in the following places:

- line 230 and 920 in the file `beacon-chain/node/node.go` .

[Fasttoken - 06/09/2023] :
The team resolved the finding by correcting the above typos in the commit `88551682018d09cf69ab604d8ccb42e7024564eb` .

[CertiK - 07/06/2023] :
The variable `disableStakinContractCheck` should be **`disableStakingContractCheck`** in the following places of the commit `8198a02d28dee2b7485610279bcf24e4f0a2bf54` :

1. line 202 and 203 in the file `config/features/config.go` ;
2. line 89 and 169 in the file `config/features/flags.go` ;

[Fasttoken - 07/20/2023] :
The team heeded the advice and resolved the finding by correcting the aforementioned typos in the commit `a98c0cb06842a9032f479b27757a1d99c39327ec` .

3B8-01 | DISCUSSION ON VALUE OF `SigmoidLimit`

Category	Severity	Location	Status
Logical Issue	● Informational	beacon-chain/core/altair/sync_committee.go (consensus): 154; beacon-chain/core/helpers/validators.go (consensus): 405; config/params/testnet_fastex_chain_config.go (consensus): 33	● Resolved

Description

Files:

- `beacon-chain/core/altair/sync_committee.go`
- `beacon-chain/core/helpers/validators.go`
- `config/params/testnet_fastex_chain_config.go`

Commit:

- `3b8da2895d7067405b54c0829eee7e044a0f978e`

The block producer and sync committee member selection inherits the algorithm from the RANDAO randomness generation in the Ethereum Proof of Stake.

In Ethereum Proof of Stake, the selection is performed through a shuffle to make the list of active validators randomly, then for each validator, a random number `rand` is generated between 0 and `MaxRand` to check if the inequality

$\frac{s_i}{s} \geq \frac{rand}{MaxRand}$
holds, where s_i is the effective balance of the validator i and s is the max effective balance.

In the Fasttoken, the same approach is adopted with the following modification in the inequality

$$(2 \cdot \frac{1}{1+e^{-1.5 \cdot \frac{P_i}{P}}} - 1) \cdot \frac{s_i}{s} \geq \frac{rand}{MaxRand} \cdot 0.62$$

The sigmoid function on the left is used to adjust the effective balance of the validator, where the P_i is the voting power of the validator i and P is the max voting power of all the validators.

The current value of `SigmoidLimit` is **0.62** on the right, but the maximum value of the sigmoid function $2 \cdot \frac{1}{1+e^{-1.5 \cdot \frac{P_i}{P}}} - 1$ is around **0.635** when the P_i equals to P.

In this case, if the voting power of validator i, $P_i = P * 97\%$, then the value of the sigmoid is **0.62**. That means a validator only needs 97% (not 100%) of the maximum voting power P to obtain the same formula as the Ethereum Proof of Stake.

Recommendation

The auditing team would like to understand the intention to choose a different implementation for a different version and wants to confirm if the two implementations are flipped.

Alleviation

[Fasttoken - 05/25/2023] :

The team removed the logic related to the sigmoid function, which makes the finding obsolete. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

BEA-01 | TYPO IN ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	● Informational	beacon-chain/core/blocks/activities.go (33b75d4): 26, 39; beacon-chain/core/fastex-phase1/attestation.go (33b75d4): 55; beacon-chain/node/node.go (33b75d4): 229, 239; beacon-chain/p2p/pubsub_filter.go (33b75d4): 57; beacon-chain/rpc/apimiddleware/custom_hooks.go (33b75d4): 849	● Resolved

Description

Files:

- `beacon-chain/core/blocks/activities.go`
- `beacon-chain/core/fastex-phase1/attestation.go`
- `beacon-chain/p2p/pubsub_filter.go`
- `beacon-chain/rpc/apimiddleware/custom_hooks.go`
- `beacon-chain/node/node.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

There are some typos in the error messages in the current codebase:

beacon-chain/core/blocks/activities.go

- in line 26, `activties` should be `activities` ;
- in line 39, `core.ProcessActivtiyNoVerifySignature` should be `core.ProcessActivityNoVerifySignature` .

beacon-chain/core/fastex-phase1/attestation.go

- in line 55, `altair.ProcessAttestationNoVerifySignature` should be `fastexphase1.ProcessAttestationNoVerifySignature` ;

beacon-chain/p2p/pubsub_filter.go

- in line 57, `Could not determine Bellatrix fork digest` should be `Could not determine fastexPhase1 fork digest` .

beacon-chain/rpc/apimiddleware/custom_hooks.go

- in line 849, `4 unsupported block version '%s'` should be `unsupported block version '%s'`.

beacon-chain/node/node.go

- in line 229, `Registering Determinstic Genesis Service` should be `Registering Deterministic Genesis Service`;
- in line 239, `Registering Intial Sync Service` should be `Registering Initial Sync Service`.

Recommendation

Recommend correcting the aforementioned typos to improve the code readability.

Alleviation

[Fasttoken - 05/25/2023] :

The team heeded the advice and resolved the finding by either removing the relevant code or correcting the typo. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8`.

[CertiK - 05/25/2023] :

`Determinstic` has not been corrected in the following code of file **beacon-chain/node/node.go**:

```
226     log.Debugln("Registering Determinstic Genesis Service")
227     if err := beacon.registerDeterminsticGenesisService(); err != nil {
228         return nil, err
229     }
```

[Fasttoken - 07/06/2023] :

The team resolved the finding by correcting the aforementioned typo in the commit `8198a02d28dee2b7485610279bcf24e4f0a2bf54`.

COB-02 | DISCUSSION ON THE USE OF THE SIGMOID FUNCTION IN BLOCK PROPOSER AND SYNC COMMITTEE MEMBERS SELECTION

Category	Severity	Location	Status
Logical Issue	● Informational	beacon-chain/core/altair/sync_committee.go (33b75d4): 143~166; beacon-chain/core/helpers/validators.go (33b75d4): 431~454	● Resolved

Description

Files:

- `beacon-chain/core/helpers/validators.go`
- `beacon-chain/core/altair/sync_committee.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

According to the current codebase, the block producer and sync committee member selection inherits the algorithm from the RANDAO randomness generation in the Ethereum Proof of Stake. Before the version `FastexPhase1` , the implementation contains some modifications that use the sigmoid function and validator's power defined by the validator's activity score.

In Ethereum Proof of Stake, the selection is performed through a shuffle to make the list of active validators randomly, then for each validator, a random number `rand` is generated between 0 and `MaxRand` to check if the inequality

$\frac{s_i}{s} \geq \frac{rand}{MaxRand}$
holds, where s_i is the effective balance of the validator i and s is the max effective balance.

In the Fasttoken, the same approach is adopted with the following changes in the inequality

$$(2 \cdot \frac{1}{1+e^{-1.5 \cdot \frac{P_i}{P}}} - 1) \cdot \frac{s_i}{s} \geq \frac{rand}{MaxRand} \cdot 0.62$$

The sigmoid function on the left is used to adjust the effective balance of the validator, where the P_i is the power of the validator i and P is the max power of all the validators.

Scenario

Consider the following scenario:

1. Based on the design, a block proposer could possibly get 1/8 of the block reward, that is, 1/8 (=**0.125**) of `A+T` , where `A` is the gas consumed in contracts associated with validators and `T` is the transaction constant gas usage;

2. For simplicity, assume the usage is the same for every block and there is no gas consumed in contracts not associated with validators.
3. Taking the ratio 1/8 as a benchmark, we assume that a validator **X** takes 1/8 of the total power among all validators and the rest of the validators hold the remaining 7/8 of total power;
4. Assume the validator **X** that holds 1/8 of the total power is the one of max power;
5. Assume there are 10000 active validators, and the 9999 validators have the same power, $P_i = 7/(8 * 9999)$;
6. Then the ratio $\frac{P_i}{P} = \frac{7}{9999}$, which gives us the value of the sigmoid function on the left, **0.0005**;
7. Dividing this value by 0.62 is around **0.0008**;
8. Assume the effective balances of all validators are the max effective balance. In this case, the validator **X** has the probability to be selected as a block proposer is around $1/10000/(0.0008 * 9999/10000 + 1/10000)$. The result is **0.111**, which is slightly less than 0.125;
9. The concern is that this design will bring more centralization risk in the consensus as opposed to the original Ethereum Proof of Stake. It is difficult for a validator to control 1/8 of total staking, but it will be easier when combined with the activity score.

Recommendation

The auditing team would like to confirm with the Fasttoken team the possible scenario in which the validator of max power could have too much power in the consensus upon the introduction of activity score and power.

Alleviation

[Fasttoken - 05/25/2023] :

The team removed the logic related to the sigmoid function, which makes the finding obsolete. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

COB-03 | DISCUSSION ON TWO IMPLEMENTATIONS OF BLOCK PROPOSER AND SYNC COMMITTEE SELECTION IN DIFFERENT VERSIONS

Category	Severity	Location	Status
Logical Issue	● Informational	beacon-chain/core/altair/sync_committee.go (33b75d4): 64~68; beacon-chain/core/helpers/beacon_committee.go (33b75d4): 452~464	● Resolved

Description

Files:

- `beacon-chain/core/helpers/beacon_committee.go`
- `beacon-chain/core/helpers/validators.go`
- `beacon-chain/core/altair/sync_committee.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

In both functions `BeaconProposerIndex()` and `precomputeProposerIndices()`, an `if-else` logic is coded to select the function for computing the index of the proposer based on the `Version`.

The function `BeaconProposerIndex()` in **beacon-chain/core/helpers/validators.go**:

```
    if state.Version() < version.FastexPhase1 {
        return ComputeProposerIndex(state, indices, seedWithSlotHash)
    }
    return ComputeProposerIndexFastexPhase1(state, indices, seedWithSlotHash)
```

The function `precomputeProposerIndices()` in **beacon-chain/core/helpers/beacon_committee.go**:

```
        if state.Version() < version.FastexPhase1 {
            index, err = ComputeProposerIndex(state, activeIndices,
seedWithSlotHash)
            if err != nil {
                return nil, err
            }
        } else {
            index, err = ComputeProposerIndexFastexPhase1(state, activeIndices,
seedWithSlotHash)
            if err != nil {
                return nil, err
            }
        }
        proposerIndices[i] = index
```

Based on our understanding, when the version is before the `FastexPhase1` , the consensus client should still be in `PoS` mode, and the logic for calculating the index of the proposer should be the same as in Ethereum. However, in the implementation, the `ComputeProposerIndex()` function takes not only the **effective balance** of the validator but also the **effective activity**, which should be calculated in the `PoSA` mode.

The function `ComputeProposerIndex()` in **beacon-chain/core/helpers/validators.go**:

```
// ComputeProposerIndex returns the index sampled by effective balance, which is
used to calculate the proposer.

func ComputeProposerIndex(bState state.ReadOnlyBeaconState, activeIndices []
primitives.ValidatorIndex, seed [32]byte) (primitives.ValidatorIndex, error) {
    length := uint64(len(activeIndices))
    if length == 0 {
        return 0, errors.New("empty active indices list")
    }
    maxRandomByte := new(big.Float).SetUint64(1<<16 - 1)
    hashFunc := hash.CustomSHA256Hasher()

    txGasPerPeriod := bState.TransactionsGasPerPeriod()
    var nonStakersGasPerPeriod uint64
    // Ignore nonStakersGasPerPeriod in post-FastexPhase1 fork.
    if bState.Version() < version.FastexPhase1 {
        nonStakersGasPerPeriod = bState.NonStakersGasPerPeriod()
    }
...
```

In addition, a similar scenario occurs in the sync committee members selection:

beacon-chain/core/altair/sync_committee.go

```go
func NextSyncCommittee(ctx context.Context, s state.BeaconState) (*ethpb.
SyncCommittee, error) {
    var indices []primitives.ValidatorIndex
    var err error
    if s.Version() < version.FastexPhase1 {
        indices, err = NextSyncCommitteeIndices(ctx, s)
    } else {
        indices, err = NextSyncCommitteeIndicesFastexPhase1(ctx, s)
    }
```

Before the `FastexPhase1`, the `NextSyncCommittee()` calls the `NextSyncCommitteeIndices()` that needs the activity score to compute the validator power:

beacon-chain/core/altair/sync_committee.go

```go
func NextSyncCommitteeIndices(ctx context.Context, s state.BeaconState) ([]
primitives.ValidatorIndex, error) {
    epoch := coreTime.NextEpoch(s)
    indices, err := helpers.ActiveValidatorIndices(ctx, s, epoch)
    if err != nil {
        return nil, err
    }
    seed, err := helpers.Seed(s, epoch, params.BeaconConfig().
DomainSyncCommittee)
    if err != nil {
        return nil, err
    }
    count := uint64(len(indices))
    cfg := params.BeaconConfig()
    syncCommitteeSize := cfg.SyncCommitteeSize
    cIndices := make([]primitives.ValidatorIndex, 0, syncCommitteeSize)
    hashFunc := hash.CustomSHA256Hasher()

    txGasPerPeriod := s.TransactionsGasPerPeriod()
    var nonStakersGasPerPeriod uint64
    // Ignore nonStakersGasPerPeriod in post-FastexPhase1 forks.
    if s.Version() < version.FastexPhase1 {
        nonStakersGasPerPeriod = s.NonStakersGasPerPeriod()
    }
...
```

In the post `FastexPhase1`, it invokes the `NextSyncCommitteeIndicesFastexPhase1()` that only utilizes the **effective balance**:

```
func NextSyncCommitteeIndicesFastexPhase1(ctx context.Context, s state.BeaconState) ([]primitives.ValidatorIndex, error) {
    epoch := coreTime.NextEpoch(s)
    indices, err := helpers.ActiveValidatorIndices(ctx, s, epoch)
    if err != nil {
        return nil, err
    }
    seed, err := helpers.Seed(s, epoch, params.BeaconConfig().DomainSyncCommittee)
    if err != nil {
        return nil, err
    }
    count := uint64(len(indices))
    cfg := params.BeaconConfig()
    syncCommitteeSize := cfg.SyncCommitteeSize
    cIndices := make([]primitives.ValidatorIndex, 0, syncCommitteeSize)
    hashFunc := hash.CustomSHA256Hasher()

    for i := primitives.ValidatorIndex(0); uint64(len(cIndices)) < params.BeaconConfig().SyncCommitteeSize; i++ {
        if ctx.Err() != nil {
            return nil, ctx.Err()
        }
...
```

Recommendation

The auditing team would like to understand the intention to choose a different implementation for a different version and wants to confirm if the two implementations are flipped.

Alleviation

[Fasttoken - 05/25/2023] :

The team only kept one implementation by removing other implementations based on the versions. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

COE-03 | INCONSISTENCY BETWEEN IMPLEMENTATION AND WHITEPAPER

Category	Severity	Location	Status
Logical Issue	● Informational	beacon-chain/core/altair/sync_committee.go (consensus): 138; beacon-chain/core/helpers/validators.go (consensus): 389	● Resolved

Description

Files:

- `beacon-chain/core/altair/sync_committee.go`
- `beacon-chain/core/helpers/validators.go`

Commit:

- `3b8da2895d7067405b54c0829eee7e044a0f978e`

According to the Fasttoken consensus whitepaper, the voting power of validator i is defined as:

$$P_i^e = T_e \cdot \frac{s_i}{S} + A_{V_i}^e,$$

where T_e is the sum of all transaction constant gas usage components, that is, $T_e = 21000 * N$ where N is the number of all transactions which have been executed during the epoch e.

However, in line 389 of the implementation:

beacon-chain/core/helpers/validators.go, **beacon-chain/core/altair/sync_committee.go**

```go
func ComputeProposerIndex(bState state.ReadOnlyBeaconState, activeIndices []
primitives.ValidatorIndex, seed [32]byte) (primitives.ValidatorIndex, error) {
    length := uint64(len(activeIndices))
    if length == 0 {
        return 0, errors.New("empty active indices list")
    }
    maxRandomByte := new(big.Float).SetUint64(1<<16 - 1)
    hashFunc := hash.CustomSHA256Hasher()

    txGasPerPeriod := bState.TransactionsGasPerPeriod()
    var nonStakersGasPerPeriod uint64
    // Ignore nonStakersGasPerPeriod in post-FastexPhase1 fork.
    if bState.Version() < version.FastexPhase1 {
        nonStakersGasPerPeriod = bState.NonStakersGasPerPeriod()
    }
    totalBalance := TotalBalance(bState, activeIndices)
    maxPower, err := MaxPower(bState, activeIndices, totalBalance,
 txGasPerPeriod, nonStakersGasPerPeriod)
    maxPowerFloat := new(big.Float).SetInt(maxPower)
    if err != nil {
        return 0, err
    }

    for i := uint64(0); ; i++ {
        candidateIndex, err := ComputeShuffledIndex(primitives.ValidatorIndex(i
%length), length, seed, true /* shuffle */)
        if err != nil {
            return 0, err
        }
        candidateIndex = activeIndices[candidateIndex]
        if uint64(candidateIndex) >= uint64(bState.NumValidators()) {
            return 0, errors.New("active index out of range")
        }
        b := append(seed[:], bytesutil.Bytes8(i/16)...)
        hash := hashFunc(b)
        bytes2 := append([]byte{}, hash[i%16], hash[16+i%16])
        randomBytes := new(big.Float).SetUint64(uint64(bytesutil.FromBytes2(
bytes2)))
        v, err := bState.ValidatorAtIndexReadOnly(candidateIndex)
        if err != nil {
            return 0, err
        }

        totalBalanceBig := new(big.Int).SetUint64(totalBalance / params.
BeaconConfig().EffectiveBalanceIncrement)
        effectiveBalanceBig := new(big.Int).SetUint64(v.EffectiveBalance() /
 params.BeaconConfig().EffectiveBalanceIncrement)
        effectiveActivityBig := new(big.Int).SetUint64(v.EffectiveActivity())
        txGasBig := new(big.Int).SetUint64(txGasPerPeriod)
        nonStakersGasBig := new(big.Int).SetUint64(nonStakersGasPerPeriod)

        var power *big.Int
        power = new(big.Int).Add(txGasBig, nonStakersGasBig)
        power = new(big.Int).Mul(power, effectiveBalanceBig)
        power = new(big.Int).Div(power, totalBalanceBig)
        power = new(big.Int).Add(power, effectiveActivityBig)

...
```

$T_e = T + B$, where T is the aggregation of the constant gas usage argument equal to 21000, and B is the gas usage of smart contracts not associated with any validator, both of which are calculated in the sliding window of 1575 epochs, not in each epoch.

The B part, `nonStakersGasBig` is zero only in the post-FastexPhase1 fork (shown in lines 354-356), which matches the formula in the whitepaper.

Recommendation

Recommend adjusting the description in the whitepaper to align with the implementation if this is the intended implementation.

Alleviation

[Fasttoken - 05/25/2023] :

The team resolved the finding by removing the `nonStakersGasPerPeriod` (B) from the implementation and difference among the versions. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8`.

[CertiK - 05/25/2023] :

The inconsistency of the notation `$T_e$` on the documentation and implementation has been consolidated into another finding.

DEO-02 | DISCUSSION ON CONTRACT REGISTRATION WITH VALIDATORS

Category	Severity	Location	Status
Logical Issue	● Informational	beacon-chain/core/blocks/deposit.go (consensus): 172	● Resolved

Description

Files:

- `beacon-chain/core/blocks/deposit.go`

Commit:

- `3b8da2895d7067405b54c0829eee7e044a0f978e`

The Fasttoken consensus algorithm utilizes the gas usage of contracts that are associated with the validators as activity scores to compute the power of the validators.

According to the current codebase, the registration of contracts with validators occurs in the contract `deposit_contract` in the execution layer where the validators are able to deposit the stake as well as the contract address for the registration process.

In the current implementation, the validator only needs to pass an address for the registration. The auditing team would like to confirm with the Fasttoken team if the following points are taken into account:

1. The passed address has not been validated that is associated with an existing contract, which means it could be an EOA or a placeholder for future contract deployment. Since some classes of addresses (i.e., vanity addresses) are popular in the blockchain, the validators could register many such addresses. If some contract is deployed in the future with these addresses, the validator will own the activity generated by these contracts.
2. If the passed address comes from an existing contract, it could belong to other deployers and not necessarily be owned by this validator. Considering the blockchain is a dark forest, the contract address registration could also be front run by other validators.

Recommendation

The auditing team would like to confirm with the Fasttoken team if these scenarios have been considered.

Alleviation

[Fasttoken - 06/06/2023] :

The team resolved the finding by adding the contract deployment logic in the execution layer in the commit `716ea69939139eab9f45b4c68347eb67de492bea` and changed the corresponding logic in the deposit contract in the consensus layer in the commit `cffbd04e743737989e44cf0ebae70fd353c5a539` .

DES-02 | DISCUSSION ON INCONSISTENCY BETWEEN DEPOSIT CONTRACT AND ITS BINDING

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/deposit/contract.go (consensus): 269~279; contracts/deposit/deposit_contract.sol (consensus): 101~106	● Resolved

Description

Files:

- `contracts/deposit/contract.go`
- `contracts/deposit/deposit_contract.sol`

Commit:

- `3b8da2895d7067405b54c0829eee7e044a0f978e`

The contract `deposit_contract` serves as the entry point for the validator registration on the execution layer. In the current codebase, the `deposit_contract` has not been modified to align with Fasttoken's new design. Its binding `contract.go` seems to be updated as the ABI is different from the current `deposit_contract` , but the deposit event is not updated, as it does not contain `DeployedAddress` and `DeploymentNonce` :

contracts/deposit/contract.go

```go
type DepositContractDepositEvent struct {
    Pubkey                []byte
    WithdrawalCredentials []byte
    Amount                []byte
    Signature             []byte
    Index                 []byte
    Raw                   types.Log // Blockchain specific contextual infos
}
```

Additionally, the function `Deposit()` is supposed to be only used for testing as it includes hardcoded address and nonce:

contracts/deposit/contract.go

```
func (_DepositContract *DepositContractTransactor) Deposit(opts *bind.
TransactOpts, pubkey []byte, withdrawal_credentials []byte, signature []byte,
 deposit_data_root [32]byte) (*types.Transaction, error) {
	return _DepositContract.contract.Transact(
		opts,
		"deposit",
		pubkey,
		withdrawal_credentials, signature,
		deposit_data_root,
		common.HexToAddress("0x1111111111111111111111111111111111111111"),
		big.NewInt(1))
}
```

Recommendation

The auditing team wants to confirm with the Fasttoken team if the updated `deposit_contract` could be provided and if the `contract.go` reflects the latest changes.

Alleviation

[Fasttoken - 06/06/2023] :

The team resolved the finding by updating the deposit contract and its binding files in the commit `cffbd04e743737989e44cf0ebae70fd353c5a539`.

GLOBAL-01 | CURRENT VERSION DOES NOT CONTAIN PATCH FOR MEV-BOOST ATTACK

Category	Severity	Location	Status
Inconsistency	● Informational		● Resolved

Description

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

MEV bots serves as a tool to frontrun a pending transaction to extract the value. To prevent being frontrun by themselves, MEV bots could use MEV-Boost/Relays as trusted mediator between block producers and block builders, which is an implementation of proposer-builder separation (PBS) built by Flashbots for the Ethereum Proof of Stake.

Validators could run MEV-Boost to maximize their staking reward by selling blockspace to an open market of builders. Block proposers could bid on transactions, then builders create blocks that include the most valuable transactions, and validators sign the transactions. Normally, blocker proposers can not see the transactions in the block until they signed the block header, which makes it difficult to frontrun the transactions in the block.

To identify transactions for exploit, the validator sent a signed, invalid block to MEV-Boost/Relay, which replied with the transactions that should have been included in that block. With the transactions in the block revealed, the validator could observe the transactions and manipulate the transactions. This critical vulnerability was exploited on April 3rd, 2023, which leads to ~20M asset loss of multiple sandwich bots.

The patch has been released on the `MEV-Boost Relay` , but it requires the corresponding modification of the client, which has been released in the Prysm v4.0.2, but it is not included in the Prysm v3.2.2.

Since the Bahamut is built on the Prysm v3.2.2 and the validators may also run the MEV-Boost/Relay, it is recommended to upgrade to the latest version to ensure the fix work properly.

Reference:

- `Post mortem`
- `MEV Bot Incident Analysis`

Recommendation

Recommend updating to the latest version (or at least v4.0.2) to include the patch.

Alleviation

[Fasttoken - 05/25/2023] :

The team heeded the advice and resolved the finding by updating the codebase to Prysm v4.0.3. The change is reflected in the commit `3226f8330911cb8df77e775f0155b335ba771bd8` .

REW-01 | DISCUSSION ON THE CALCULATION OF `BaseProposerReward`

Category	Severity	Location	Status
Logical Issue	● Informational	beacon-chain/core/fastex-phase1/reward.go (33b75d4): 50	● Resolved

Description

Files:

- `beacon-chain/core/fastex-phase1/reward.go`

Commit:

- `33b75d4e162179d360e60ac88bb4289293b530a6`

The fasttoken introduces a novel proposer base reward calculation based on the validator's power via the activity score associated with the validator's registered contracts. The function `BaseProposerReward()` is used to compute the base proposer reward as follows:

$$BPR = \frac{(A+T)*bf}{W*n*gwei},$$

where $A = \sum_{i=1}^{n} ea_i$ is the total effective activities of the n validators and T is the transaction constant gas in the window of 1575 epochs. The bf is the base fee of the block and W is the epoch window size 1575, n is the number of validators and $gwei$ is the constant 10^9.

Therefore, the base proposer reward in each epoch is $BPR * s$, where s (= 32) is the number of slots in an epoch.

On the other hand, the total validator base reward per epoch is given by:

$BR_{total} = \frac{B*f}{\sqrt{B}} = f * \sqrt{B}$, where B is the total active balance, and f is the constant 156.

Assume that the current number of validators is the target number 4096, and each of them has an effective balance of $8192 * 10^9$. Then the base reward is

$$156 * \sqrt{4096 * 8192 * 10^9} = 0.9 * 10^9 gwei.$$

According to the design, $7/8$ of the reward will be distributed to the validators for attestation rewards and participating sync committees. In the Ethereum PoS, the remaining $1/8$ of the reward (i.e., $0.1125 * 10^9$) is granted to the block proposers for proposing blocks.

The fasttoken attempts to use the base proposer reward in each epoch (i.e., $32 * BPR$) to replace the $1/8$ of the total validator base reward per epoch as the reward to the block proposers. In this case, assume that each block has a half load (15M gas consumed) and the base fee is $100 * gwei$, then the base proposer reward in each epoch is

$$32 * \frac{(A+T)*bf}{W*n*gwei} = 32 * \frac{(32*15M)*100*gwei}{4096*gwei} = 0.395 * 10^9 gwei,$$

which is larger than the $1/8$ of the total validator base reward per epoch ($0.1125 * 10^9$). In this case, the block proposers will be incentivized for their duties.

However, the auditing team has the following points that would like to check with the fasttoken team:

1. The base proposer reward BPR depends on the $A + T$. If the block activity is low for a long time, then the BPR could be very small (because it is linear with $A + T$.) compared to the $1/8$ stable reward;
2. If the number of validators increases, then the BR_{total} increases but BPR decreases. Take the max number of validators, 20480 as an example, the BPR will be $1/5$ of the previous one, which is $0.079 * 10^9$ but the $1/8$ of the total validator base reward is $0.25 * 10^9$.

In both cases, the block proposer reward based on the new design could be less than the $1/8$ of the total validator base reward in the old design. As a result, the block proposers could possibly be disincentivized to participate in the consensus because the reward in the new design is not predictable and prone to change.

Recommendation

The auditing team would like to check with the fasttoken team if this is the intended design.

Alleviation

[Fasttoken - 07/07/2023] :

The team confirmed that this is the intended design. The base proposer reward is the average burned amount of FTNs in a block during period.

STF-01 | TYPO IN THE CODEBASE OF EXECUTION LAYER

Category	Severity	Location	Status
Coding Style	● Informational	core/vm/stateful_contracts.go (execution-716ea69): 65, 70, 102	● Resolved

Description

Files:

- `core/vm/stateful_contracts.go`

Commit:

- `716ea69939139eab9f45b4c68347eb67de492bea`

In the aforementioned places, `statefulPrecomiledContractWithSelectors` should be `statefulPrecompiledContractWithSelectors` .

Recommendation

Recommend correcting the typo to improve the code readability.

Alleviation

[Fasttoken - 07/06/2023] :

The team heeded the advice and resolved the finding in the commit `3d669ac92faa0747a2aa2e8905e46d39c563d114` .

STT-02 | TYPO IN THE CODEBASE OF CONSENSUS LAYER

Category	Severity	Location	Status
Coding Style	● Informational	beacon-chain/state/stateutil/activity_changes_root.go (consensus-cffbd04): 17, 23; beacon-chain/state/stateutil/field_root_shared_activity.go (consensus-cffbd04): 15; beacon-chain/state/stateutil/shared_activity_root.go (consensus-cffbd04): 13	● Resolved

Description

Files:

- `beacon-chain/state/stateutil/activity_changes_root.go`
- `beacon-chain/state/stateutil/field_root_shared_activity.go`
- `beacon-chain/state/stateutil/shared_activity_root.go`

Commit:

- `cffbd04e743737989e44cf0ebae70fd353c5a539`

beacon-chain/state/stateutil/activity_changes_root.go

- in lines 17 and 23, `merkleiztion` should be `merkleization`.

beacon-chain/state/stateutil/field_root_shared_activity.go

- in line 15, function name `SharedActivityRootWithHaher()` should be `SharedActivityRootWithHasher()`.

beacon-chain/state/stateutil/shared_activity_root.go

- in line 13, function name `SharedActivityRootWithHaher()` should be `SharedActivityRootWithHasher()`.

Recommendation

Recommend correcting the typo to improve the code readability.

Alleviation

[Fasttoken - 07/06/2023] :

The team heeded the advice and resolved the finding in the commit `8198a02d28dee2b7485610279bcf24e4f0a2bf54`.

VAL-02 | TYPO IN FUNCTION NAME `isEligibileForActivationQueue()`

Category	Severity	Location	Status
Coding Style	● Informational	beacon-chain/core/helpers/validators.go (3226f83): 498, 504, 508	● Resolved

Description

Files:

- `beacon-chain/core/helpers/validators.go`

Commit:

- `3226f8330911cb8df77e775f0155b335ba771bd8`

In the line 498, 504 and 508 of file `beacon-chain/core/helpers/validators.go` , the function name `isEligibileForActivationQueue()` is supposed to be `isEligibleForActivationQueue()` .

Recommendation

Recommend correcting the typo to improve the code readability.

Alleviation

[Fasttoken - 07/06/2023] :

The team heeded the advice and resolved the finding in the commit `8198a02d28dee2b7485610279bcf24e4f0a2bf54` .

VAL-03 | CODE SIMPLIFICATION IN FUNCTION `RandomBytes()`

Category	Severity	Location	Status
Coding Style	● Informational	beacon-chain/core/helpers/validators.go (3226f83): 422~424, 430~434	● Resolved

Description

Files:

- `beacon-chain/core/helpers/validators.go`

Commit:

- `3226f8330911cb8df77e775f0155b335ba771bd8`

The function `RandomBytes()` is intended to generate a pseudo-random number between 0 and `totalEffectivePower` -1. The `randomNumber` generated in line 424 with index 0 will be overwritten by the for loop in line 430, which can be merged into the for loop and start the index with 0.

```
419 func RandomBytes(seed [32]byte, totalEffectivePower uint64) uint64 {
420     maxRandomBytes := uint64(1<<64 - 1)
421     hashFunc := hash.CustomSHA256Hasher()
422     hash := hashFunc(append(seed[:], bytesutil.Bytes8(0)...))
423     randomBytes := hash[:8]
424     randomNumber := bytesutil.FromBytes8(randomBytes)
425 
426     if totalEffectivePower == 0 {
427         return 0
428     }
429 
430     for i := uint64(1); randomNumber > (maxRandomBytes/totalEffectivePower)*
totalEffectivePower; i++ {
431         hash = hashFunc(append(seed[:], bytesutil.Bytes8(i)...))
432         randomBytes = hash[:8]
433         randomNumber = bytesutil.FromBytes8(randomBytes)
434     }
435 
436     return randomNumber % totalEffectivePower
437 }
```

Recommendation

Recommend merging the random number generation with index 0 into the for loop.

Alleviation

[Fasttoken - 07/20/2023] :

The team heeded the advice and resolved the finding by merging the random number generation with index 0 into the for loop in the commit `a98c0cb06842a9032f479b27757a1d99c39327ec` .

```
  for i := uint64(0); ; i++ {
        hash := hashFunc(append(seed[:], bytesutil.Bytes8(i)...))
        random = bytesutil.FromBytes8(hash[:8])
        if random <= (maxRandomBytes/totalEffectivePower)*totalEffectivePower {
            return random % totalEffectivePower
        }
    }
```

[CertiK - 07/20/2023] :

Recommend changing the `<=` to `<` in the following condition so that the returned values in `[0, totalEffectivePower)` have the same probability:

```
        if random <= (maxRandomBytes/totalEffectivePower)*totalEffectivePower {
            return random % totalEffectivePower
        }
```

[Fasttoken - 07/25/2023] :

The team heeded the advice and resolved the finding by changing `<=` to `<` in the commit `b7e967722abcf62356caaf0c20e536f3746e41b8` .

VAL-04 | INCONSISTENCY BETWEEN IMPLEMENTATION AND WHITEPAPER ON THE CALCULATION OF VALIDATOR'S POWER

Category	Severity	Location	Status
Inconsistency	● Informational	beacon-chain/core/helpers/validators.go (3226f83): 370, 454	● Acknowledged

Description

Files:

- `beacon-chain/core/helpers/validators.go`

Commit:

- `3226f8330911cb8df77e775f0155b335ba771bd8`

According to the Fasttoken consensus whitepaper, the i-th validator's (denoted as V_i) power is defined as:

$$P_i^e = T_e \cdot \frac{s_i}{S} + A_{V_i}^e,$$

where

- T_e is the sum of all transaction constant gas usage components, that is, $T_e = 21000 * N$ where N is the number of all transactions which have been executed during the epoch e.
- s_i is the staked amount of the i-th validator;
- S is the sum of all validators' staked balances;
- $A_{V_i}^e$ is the activity score assigned to the validator V_i for the epoch e.

In the implementation, the $\frac{s_i}{S}$ is assumed that the staked amounts of all the validators are the equal, so it is $\frac{1}{n}$ (n is the number of active validators):

```
transactionsGas := sharedActivity.TransactionsGasPerPeriod / length
```

Moreover, the T_e and $A_{V_i}^e$ is the rolling sum of window size `period` given by the formula

```
effectiveActivity := ((val.EffectiveActivity+activity)*period -
val.EffectiveActivity) / period
```

and

```
sharedActivity.TransactionsGasPerPeriod = ((gasPerPeriod+gasPerEpoch)*period -
gasPerPeriod) / period
```

, which are not the values in the current epoch e.

Recommendation

Recommend adjusting the whitepaper if the implementation is the intended design.

Alleviation

[Fasttoken - 07/20/2023] :

Issue acknowledged. The team will fix the issue in the future, which will not be included in this audit engagement.

APPENDIX | BAHAMUT EXECUTION AND CONSENSUS

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | **Securing** the **Web3** World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.