ANCILIA

Money Market Fund Auditing Report

v1.4

May 2024

Prepared for

Franklin Templeton

Prepared by

Ancilia, Inc.

Index

Version History

Version	Description	Date
v1.4	Reviewed comment update	05/13/2024
v1.3	FT-A-24 update	05/10/2024
v1.2	Status update	05/07/2024
v1.1	Feedbacks and status update	04/30/2024
v1.0	Initial report	04/26/2024
Draft 3	Technique draft #3	04/25/2024
Draft 2	Technique draft #2	04/22/2024
Draft 1	Technique draft #1	04/20/2024

Executive Summary

The Franklin Templeton team (FT) shared their smart contract source code in an archive via Microsoft Teams. We have listed hashes of smart contracts to ensure the entirety of the audit can be tied to a given contract version. The Ancilia research team has worked with the Franklin Templeton team on all potential findings and issues. The audit scope includes checking the smart contracts for attack vulnerabilities such as re-entry attacks, logic flaws, authentication bypasses, DoS attacks, etc. There are multiple versions of the implementation contracts for the same UUPS proxy and we have gone through all versions.

The Operational Security

To ensure the security of the entire project, not only does the smart contract(s) code need to be secured, but also the day to day operations.
For example,

- Contract upgrade
- Authorized user management
- Arbitrary assets movement
- Multisig user management
- Important function parameters. E.G., The price in the endOfDay() function for the dividends.

If an EOA is used as a super admin then it must be fully protected and secured. We recommend using a distributed key-generation and signature-generation schema. For example, using a Threshold Signature Schema[1] that can achieve:

- Elimination of one single private key, replaced by multiple shard keys (e.g., MPC) that can provide (t,n) threshold signature.;
- Integration with a Role-Based Access Control (RBAC) system and other security checks to ensure the safety of any operation.

Multisig can help but does not support key rotation well. There must be a visual and easy way to tell what the signed data really accomplishes. Furthermore, It is important to let the signer know the impact before they sign. A protocol specific rule can apply on said processes.

[1] "Fast Multiparty Threshold ECDSA with Fast Trustless Setup", CCS'18

Disclaimer

Note that security audit services do not guarantee to find all possible security issues in the given smart contracts. A repeating code audit or incremental code audit is encouraged. Multiple audits with several auditors are recommended. Product owners are still required to have their own test cases and regular code review process. A threat intelligence system may help to discover or prevent a potential attack which can further reduce risk. Additionally, a bug bounty program for the community will help improve the security of products. Last but not least, Security is complicated! A strong smart contract does not guarantee your product is safe from all cybersecurity attacks.

Contracts overview

After compilation with Solc(version 0.8.18), there are a total of 19 smart contracts which are listed below(test, mocks and interface files are omitted) .

Contract Name	Location	SHA256
MoneyMarketFund	contracts/FT/MoneyMarketFund.sol	15e5fffa77a257d53d95967598351fe99e609f3ecd175f0040bcf1ada17df493
ModuleRegistry	contracts/FT/infrastructure/ModuleRegistry.sol	55e9abfaf3cabdb38587025a0384cfa9f018ec9409ed812d9dca7e086ae799d9
TokenRegistry	contracts/FT/infrastructure/TokenRegistry.sol	f82eab4c3c830cea4a127d469833622fa5c4e372b6a5e69d6436b1264c4f02b6
AuthorizationModule	contracts/FT/infrastructure/modules/AuthorizationModule.sol	c38939f1d8a85bef0c9cbed119e02fb94b6b64eebdd50899020f2a7c0d6ad1f5
IntentValidationModule	contracts/FT/infrastructure/modules/IntentValidationModule.sol	a913e1aa692a56c15115765fb53271cf006f1a5210521bd0533c705748fd427d
TransactionalModule	contracts/FT/infrastructure/modules/TransactionalModule.sol	d709fb07cabcbbc11144bc5b6cde6cab0eb6f2c48bb5c0965e18a9122334a5db
TransferAgentModule	contracts/FT/infrastructure/modules/TransferAgentModule.sol	933468ff32a6b761d67b1cf2bb2e416e2848b9613c0b828d86a81b837fb7adc6
AuthorizationModule_V1	contracts/FT/infrastructure/modules/upgrade_history/authorization/AuthorizationModule_V1.sol	14eff80f88d30bc3aa3e0d39c103f514cf1e2c733ef53d46eec81d02dde4f8bb
AuthorizationModule_V2	contracts/FT/infrastructure/modules/upgrade_history/authorization/AuthorizationModule_V2.sol	412a0fb3a6840b9d7e33901b60da8c45264c315e902a72745fab98281f9ca140
MoneyMarketFund_V1	contracts/FT/infrastructure/modules/upgrade_history/token/MoneyMarketFund_V1.sol	b458ea1f835159119b64227449b1bb6784dcc679944bdd108a7819cd598dadeb
MoneyMarketFund_V2	contracts/FT/infrastructure/modules/upgrade_history/token/MoneyMarketFund_V2.sol	6d40acb4958a0663d2350ebc071e50a461be08873b4a1d6fc85432f3b8428dbd

Contract Name	Location	SHA256
MoneyMarketFund_V3	contracts/FT/infrastructure/modules/upgrade_history/token/MoneyMarketFund_V3.sol	71ece106c8f964a82ab30f9c1a8c63653ef8ff77a82a44d66bed363aede54ae8
TransactionalModule_V1	contracts/FT/infrastructure/modules/upgrade_history/transactional/TransactionalModule_V1.sol	cdff221d409ab73bad5b11b6d92db52ad07cf20a630fe512f51b2e7fd4b1c3c3
TransactionalModule_V2	contracts/FT/infrastructure/modules/upgrade_history/transactional/TransactionalModule_V2.sol	37b0be70e395e3501732d22b52ab036ef5db911c8e108db07ade726a1717362d
TransactionalModule_V3	contracts/FT/infrastructure/modules/upgrade_history/transactional/TransactionalModule_V3.sol	6c1105bcb470d06351ca6c6284302fb58ff6839d9b1ebabeec39d905627fcbb6
TransferAgentModule_V1	contracts/FT/infrastructure/modules/upgrade_history/transfer_agent/TransferAgentModule_V1.sol	514c2f6e080e61a3b5eff19cff20adecbb8010e514319acc921bffa135e70543
TransferAgentModule_V2	contracts/FT/infrastructure/modules/upgrade_history/transfer_agent/TransferAgentModule_V2.sol	10a7e2c1b6f49080e4e8b45cef63787fb324f1a321a301b47e6c58fae09f2515
TransferAgentModule_V3	contracts/FT/infrastructure/modules/upgrade_history/transfer_agent/TransferAgentModule_V3.sol	749fcc110ca3301137c7dd23de236e1034855272019ca88ec9652b5135eeccf2
MultiSigGenVerifier	contracts/FT/infrastructure/multisig/MultiSigGenVerifier.sol	33e5d80654df99207c9ccc883d91c6da7e41967b33d6b4901e57f44bcce59089

The findings

Results

ID	Description	Severity	Product Impact	Status
FT-A-11	Too few user shares to burn	Medium	Medium	Won't fix
FT-A-12	RequestId and account may not be aligned	Low	Low	Fixed
FT-A-13	renounceRole does not check account status	Low	Low	Won't fix
FT-A-14	User transaction sequence may change	Medium	Low	Future enhancement
FT-A-15	'selfService' flag is missing from events	Info	Info	Won't fix

FT-A-16	MultiSigGenVerifier returns only 'String' error code as REVERT value	Info	Info	Future enhancement
FT-A-17	MultiSigGenVerifier allows bigger weight user	Low	Low	Future enhancement
FT-A-18	The front-run opportunity for the inappropriate 'date' parameter	Low	Low	Won't fix
FT-A-19	Code optimization	Info	Info	Fixed
FT-A-20	Missing events in removeAccountPostRecovery()	Info	Info	Fixed
FT-A-21	selfService transfer allows self to self	Medium	Medium	Fixed
FT-A-22	Holderlist does not refresh after recovery	Info	Info	Fixed
FT-A-23	Recover action does not check if account is frozen	Low	Low	Won't fix
FT-A-24	Unsecure UUPS contract upgrade process	High	High	Fixed

Details

FT-A-24 [High] Unsecure UUPS contract upgrade process

In the **deploy-step5-upgrades.ts** script, the new module will be upgraded by 'upgradeProxy' function but any initialize* functions will be called separately in a different transaction. This is not a secure operation because the initialization* function should be only called during the upgrade.

```
63          if (func.name.startsWith('initialize')) {
64              console.log(`Found initializer: ${func.name}`);
65              initializeName = func.name;
66          }
67      });
68
69      contract = await upgrades.upgradeProxy(contract, newContractFactory.connect(deployment.signer), {
70      });
71  }
72
73  // call initializer, if there's any
74  if (initializeName !== '') {
75      console.log(`Calling initializer: ${initializeName}`);
76      await contract[initializeName]();
77  }
78 }
```

The calldata option is allowed in upgrades.upgradeProxy() which allows the new implementation contract to be initialized. One example for the initializeP2PCapability() function when upgrading from AuthorizationModule v1 to v2, the code could be something like this:

```
        contract = await upgrades.upgradeProxy(contract,
newContractFactory.connect(deployment.signer), {
            call: {fn: "initializeP2PCapability"}
        }
```

Please refer to this document for more information.

Suggestion:

Update: Fixed. New script uses upgrades.upgradeProxy() with 'fn' attached in the "call"

FT-A-11 [Medium] Too few user shares to burn

In the contract **TransferAgentModule**, the function _handleLiquidation() burns some amount of user shares based on the new price. It is possible that the user's shares are not enough to burn because of the difference between the old price and new price. Once _handleLiquidation() fails, the caller function _processSettlements() will fail and the external function endOfDay() will fail as well. The whole batch of account executions will be discarded.

The following case will demonstrate this:

- User A has shares 100, and the current price is 5, for a total amount of 500.
- User A calls requestCashLiquidation() with the amount 480.
- Admin calls endOfDay() with a price 4.
- The function _getQuantityOfTokens() returns share 480/4 = 120

```
477     function _getQuantityOfTokens(
478         uint256 scaleFactor,
479         uint256 amount,
480         uint256 price
481     ) internal pure virtual returns (uint256) {
```

- User A does not have enough shares, so this will cause the function endofDay() to fail.

Suggestion: Check the shares number before burn.

Update: Won't fix
The price is fixed for this type of fund. In the future, to accommodate multiple types of funds, we may need to introduce a new series of smart contracts, primarily due to the constraints on code size.

FT-A-14 [Medium] User transaction sequence may change

The contract **TransactionModule** uses EnumerableSetUpgradeable to manage the account pendingTransactionsMap. The Set does not have the sequence(order) guaranteed. If the user canceled one of the transaction requests, the sequence in the Set will be changed.
The _remove() function will delete one of the values by moving the last one in the set to the deleted location(index):

```
83  function _remove(Set storage set, bytes32 value) private returns (bool) {
84      // We read and store the value's index to prevent multiple reads from the same storage slot
85      uint256 valueIndex = set._indexes[value];
86
87      if (valueIndex != 0) {
88          // Equivalent to contains(set, value)
89          // To delete an element from the _values array in O(1), we swap the element to delete with
90          // the array, and then remove the last element (sometimes called as 'swap and pop').
91          // This modifies the order of the array, as noted in {at}.
92
93          uint256 toDeleteIndex = valueIndex - 1;
94          uint256 lastIndex = set._values.length - 1;
95
96          if (lastIndex != toDeleteIndex) {
97              bytes32 lastValue = set._values[lastIndex];
98
99              // Move the last value to the index where the value to delete is
100             set._values[toDeleteIndex] = lastValue;
101             // Update the index for the moved value
102             set._indexes[lastValue] = valueIndex; // Replace lastValue's index to valueIndex
103         }
```

If an account has multiple transactions, let's say if user A created the following transactions:

1) Cash purchase 1000 (request id TX1)
2) Cash purchase 5000 (request id TX2)
3) Cash purchase 2000 (request id TX3)
4) Cash liquidation 2000 (request id TX4)

The User A is regretted step (2) and subsequently calls the function cancelSelfServiceRequest() and request id TX2 is canceled. So the new Set will be as follows:

1) Cash purchase 1000 (request id TX1)
2) Cash liquidation 2000 (request id TX4) <- moved from previous step 4)
3) Cash purchase 2000 (request id TX3)

At the endOfDay() call it may fail because TX4 requests more shares to be liquidated.

Suggestion:

Update: Future enhancement

Currently, the sequence of transactions is not crucial. However, this may change in the future, necessitating significant modifications to the logic.

FT-A-21 [Medium] selfService transfer allows self to self

In the **TransactionalModule_V3** contract, the new requestSelfServiceShareTransfer() function allows transfer of shares between two accounts. However, there is no limit for the user transfer to itself. The impact may vary. One of the cases is to prevent users from being deauthorized or frozen because of the pending transaction.

Suggestion: check source and destination if it is the same.

Update: Fixed

FT-A-12 [Low] RequestId and account may not be aligned

In the contract **TransactionalModule**, function clearTransactionStorage() will clear requestId from the transactionDetailMap and then clear account from pendingTransactionsMap. However, if requestId does not belong to the account, the clearTransactionStorage() still executes successfully but returns False. It is better to check if the requestId is owned by the account or not. A quick check for doing so is illustrated below..

Suggestion: Change the code from:

```
    delete transactionDetailMap[requestId];
    return pendingTransactionsMap[account].remove(requestId);
```

To

```
    if (pendingTransactionsMap[account].remove(requestId)) {
        delete transactionDetailMap[requestId];
        return true;
```

```
    }
    return false;
```

Update: Fixed

FT-A-13 [Low] renounceRole does not check account status

In the contract **AuthorizationModule**, function deauthorizeAccount() revokes account role from ROLE_FUND_AUTHORIZED, and it checks if the account has pending transactions and balances. However, the function renounceRole() does not follow the same principle if the role is ROLE_FUND_AUTHORIZED. The behaviors of these two functions should be consistent.
Furthermore, the event AccountDeauthorized() will need to be emitted as well.

```
116  function deauthorizeAccount(
117      address account
118  ) external virtual override onlyRole(ROLE_AUTHORIZATION_ADMIN) {
119      require(account != address(0), "INVALID_ADDRESS");
120      address txModule = modules.getModuleAddress(
121          keccak256("MODULE_TRANSACTIONAL")
122      );
123      require(txModule != address(0), "MODULE_REQUIRED_NOT_FOUND");
124      require(
125          hasRole(ROLE_FUND_AUTHORIZED, account),
126          "SHAREHOLDER_DOES_NOT_EXISTS"
127      );
128      require(
129          !ITransactionStorage(txModule).hasTransactions(account),
130          "PENDING_TRANSACTIONS_EXIST"
131      );
132      require(
133          IHoldings(tokenAddress).getShareHoldings(account) == 0,
134          "ACCOUNT_HAS_BALANCE"
135      );
136
137      _revokeRole(ROLE_FUND_AUTHORIZED, account);
138      emit AccountDeauthorized(account);
139  }
```

```
160     function renounceRole(
161         bytes32 role,
162         address account
163     )
164         public
165         virtual
166         override(AccessControlUpgradeable, IAccessControlUpgradeable)
167     {
168         if (role == ROLE_FUND_AUTHORIZED) {
169             require(
170                 hasRole(ROLE_FUND_AUTHORIZED, account),
171                 "ACCOUNT_IS_NOT_A_SHAREHOLDER"
172             );
173             require(
174                 hasRole(ROLE_AUTHORIZATION_ADMIN, _msgSender()),
175                 "CALLER_IS_NOT_AN_ADMIN"
176             );
177         } else {
178             require(
179                 account == _msgSender(),
180                 "AccessControl: can only renounce roles for self"
181             );
182         }
183
184         _revokeRole(role, account);
185     }
```

Suggestion: Check user pending transactions and emit `AccountDeauthorized` event if the role is `ROLE_FUND_AUTHORIZED`

Update: Won't fix

This functionality is designed to prevent shareholders from renouncing their roles, as this could potentially allow non-shareholders to retain shares.

FT-A-17 [Low] MultiSigGenVerifier allows bigger weight user

The function `_setupSigner()` should not allow a signer which has a bigger weight than the `HIGH` threshold. A cap should be enforced, this will ensure the multisig will have 2 signers for any privileges function calls.

Suggestion:

Update: Future enhancement

Our multisig is based on signer weights rather than the number of signatures. Additionally, we utilize Multi-Party Computation (MPC) within our signing infrastructure.

FT-A-18 [Low] The front-run opportunity for the inappropriate 'date' parameter

In the contract **TransferAgentModule**, function `settleTransactions()` and `endOfDay()` will use `'date'` to filter the transactions which need to be executed. If the 'date' is larger or equal to the current block timestamp, a user could use a front-run to benefit from the knowing price difference. They could sell or buy to increase gains.
The `'date'` parameter must be equal or less than a finalized mined block timestamp to prevent a front-run on Polygon.

Suggestion: Ensure the `'date'` parameter must be equal or less than a finalized mined block timestamp to prevent the front-run on Polygon.

Update: Won't fix
The price is fixed for this type of fund.

FT-A-23 [Low] Recover action does not check if account is frozen

In the contract **TransferAgentModule**, The two recovery functions: `recoverAccount()` and `recoverAsset()` do not check if `'to'` account is frozen or not. At this time it is unclear if 'from' should be checked in these cases.

Suggestion:
Update: Won't fix
We oversee the entire recovery process, including both account and asset recovery. Therefore, a check within the smart contract is not necessary.

FT-A-15 [Info] 'selfService' flag is missing from events

In the contract **TransactionModule**, users can create the transaction by themselves, those transactions will have a bool `'selfService'` flag to indicate the creator.

```
613         accountsWithTransactions.add(account);
614         transactionDetailMap[requestId].txType = txType;
615         transactionDetailMap[requestId].date = date;
616         transactionDetailMap[requestId].amount = amount;
617         transactionDetailMap[requestId].selfService = selfService;
618
619         emit TransactionSubmitted(account, requestId);
620     }
```

But there is no way to track if that requestId is from the user themselves or not. The event TransactionSubmitted and TransactionSettled do not have that flag.

```
402             emit TransactionSettled(
403                 account,
404                 date,
405                 uint8(txType),
406                 txId,
407                 price,
408                 (lastBalance * price) / scaleFactor,
409                 lastBalance
410             );
```

There is no easy way to tell if that transaction id is from the user or not.

Suggestion:
Update: Won't fix

There is no business need to monitor the value of this flag beyond the explicit query made through getTransactionDetail. The primary function of this flag is to internally label the transactions initiated by the shareholder, thereby enabling them to cancel these transactions at any point.

FT-A-16 [Info] MultiSigGenVerifier returns only 'String' error code as REVERT value

In the contract **MultiSigGenVerifier,** the function signedDataExecution() will call the target contract and return the error if it fails. It assumes the return is a string and decodes that accordingly. This code is inefficient and does not support other failed cases.

Suggestion: change code from:

```
if (!success) {
    assembly {
```

```
            result := add(result, 0x04)
        }
        revert(abi.decode(result, (string)));
    }
```

to:

```
    if (!success) {
        assembly {
            revert(add(result,32), mload(result))
        }
```

Update: Won't Fix (Future enhancement)
Our multi-signature contract is intentionally non-upgradeable for security purposes. Therefore, any modifications would necessitate the deployment of an entirely new contract, along with the establishment of appropriate access control in a separate transaction. Given the significant nature of this change, we plan to introduce this enhancement when a more substantial functionality upgrade is required.

FT-A-19 [Info] Code optimization

FT-A-19.1: IntentValidationModule.sol

Original code:

```
    function _removeDeviceKey(
        address account,
        uint256 deviceId
    ) internal virtual {
        require(devicesMap[account].contains(deviceId), "INVALID_DEVICE_ID");
        delete deviceKeyMap[account][deviceId];
        devicesMap[account].remove(deviceId);
        emit DeviceKeyRemoved(account, deviceId);
    }
```

Optimized code:

```
function _removeDeviceKey(
        address account,
        uint256 deviceId
    ) internal virtual {
        require(devicesMap[account].remove(deviceId), "INVALID_DEVICE_ID");
        delete deviceKeyMap[account][deviceId];
        emit DeviceKeyRemoved(account, deviceId);
    }
```

FT-A-19.2: TransferAgentModule.sol

Function _processDividents(), original code:

```
            _payDividend(account, rate, dividendShares);
            // handle very unlikely scenario if occurs
            _handleNegativeYield(account, rate, dividendShares);
```

Optimized code:

```
        require(rate != 0, "INVALID_RATE");
        if (rate > 0) {
            _payDividend(account, rate, dividendShares);
        } else {
            // handle very unlikely scenario if occurs
            _handleNegativeYield(account, rate, dividendShares);
        }
```

Suggestion:

Update: Fixed

FT-A-20 [Info] Missing events in removeAccountPostRecovery()

In the **AuthorizationModule_V2** contract, the function removeAccountPostRecovery() revokes two roles: ACCESS_CONTROL_FROZEN and ROLE_FUND_AUTHORIZED. There are no events being emitted for those two roles when revoked and it is not consistent(other functions do).
For example::
unfreezeAccount() emits AccountUnfrozen() event.
deauthorizeAccount() emits AccountDeauthorized() event.

Suggestion:

Update: Fixed

FT-A-22 [Info] Holderlist does not refresh after recovery

There is one code change from **TransferAgentModule** v2 to v3 that the holder list won't refresh after the recovery. This will help save gas cost on function _processDividends() which will check if user balance is zero.

```
296     // Effects & Interactions
297     moneyMarketFund.burnShares(from, balance);
298     moneyMarketFund.mintShares(to, balance);
299-    moneyMarketFund.updateHolderInList(from);
300-    moneyMarketFund.updateHolderInList(to);
301     IAccountManager(modules.getModuleAddress(AUTHORIZATION_MODULE))
302         .removeAccountPostRecovery(from, to);
303
304     emit AccountRecovered(from, to, balance, memo);
305 }
```

```
308     // Effects & Interactions
309     moneyMarketFund.burnShares(from, balance);
310     moneyMarketFund.mintShares(to, balance);
311     IAccountManager(modules.getModuleAddress(AUTHORIZATION_MODULE))
312         .removeAccountPostRecovery(from, to);
313
314     emit AccountRecovered(from, to, balance, memo);
315 }
```

```
427     function _processDividends(
428         address account,
429         uint256 date,
430         int256 rate,
431         uint256 price
432     ) internal virtual {
433         if (moneyMarketFund.hasHoldings(account)) {
434             uint256 dividendAmount = moneyMarketFund.balanceOf(account) *
435                 uint256(abs(rate));
436             uint256 dividendShares = dividendAmount / price;
```

Suggestion:

Update: Fixed

Summary

Ancilia team has performed both an automated and manual code audit on the Money Market Fund smart contracts mentioned above. All issues have been shared with the Franklin Templeton team through one of the Microsoft Team channels. Overall, 0 critical, 1 high, 3 medium, 5 low and 5 informational impact issues have been discovered through this audit.

Recommendations

1. Implement Ongoing Security Monitoring

Continuous monitoring of the smart contract's activity and state is crucial to promptly detect any abnormal behavior or potential security breaches. Utilize specialized tools and platforms that provide real-time monitoring and alerts for suspicious activities.

2. Adopt Preventive Security Measures

Incorporate preventive security measures within the smart contract architecture to mitigate common vulnerabilities such as reentrancy, integer overflow, and unauthorized access. Employ techniques like input validation, access control mechanisms, and secure coding practices to reduce the attack surface.

3. Utilize Multi-Party Computation (MPC) or Multisig Solutions

Enhance the security of administrative functions and sensitive operations by leveraging Multi-Party Computation (MPC) or Multisig solutions. Distributing control over critical actions among multiple parties reduces the risk of single points of failure and unauthorized access, thereby increasing the resilience against attacks targeting administrative privileges.

4. Regularly Update and Patch Smart Contracts

Stay vigilant against emerging threats and vulnerabilities by keeping the smart contract codebase up-to-date with the latest security patches and best practices. Establish a structured process for code review, vulnerability assessment, and timely deployment of updates to address newly discovered weaknesses.

5. Educate Stakeholders on Operational Security Best Practices

Educate project stakeholders, including developers, administrators, and end-users, about operational security best practices relevant to smart contract management. Foster a culture of security awareness and provide resources for training and guidance on secure deployment, configuration, and maintenance procedures.

6. Engage with Security Experts for Periodic Reviews

Engage with experienced security professionals or auditors periodically to conduct comprehensive reviews of the smart contract infrastructure and operational processes. Solicit feedback and recommendations for enhancing security posture, addressing evolving threats, and ensuring compliance with industry standards and regulatory requirements.

By incorporating these recommendations into your operational practices, you can enhance the overall security posture of the smart contract ecosystem and mitigate the risks associated with potential security vulnerabilities and threats. Remember, security is a continuous journey that requires proactive measures and ongoing vigilance to protect against emerging risks and safeguard valuable assets.