BLOCKSEC

Security Audit Report for PopDEX Bridge

Date: November 16, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

Chapter 2 Findings **5**
2.1 Security Issue . . . 5
2.1.1 Lack of role conflict checks in function `finalizeVerifierUpdate()` . . . 5
2.2 Recommendation . . . 6
2.2.1 Align custody-chain timing bounds between Hub and Spoke . . . 6
2.2.2 Add Locker availability checks in function `_setLockerBatch()` . . . 8
2.2.3 Reject the zero address in system candidate updates . . . 9
2.2.4 Ensure withdrawals remain counted for the full 24-hour window . . . 9
2.3 Note . . . 12
2.3.1 Potential centralization risks . . . 12
2.3.2 Out of scope dependencies and external logic . . . 12

Report Manifest

Item	Description
Client	PopDEX
Target	PopDEX Bridge

Version History

Version	Date	Description
1.0	November 16, 2026	First release

Signature



Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-11-16 08:09:26+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity, Rust
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of PopDEX Bridge of PopDEX.

The current update introduces additional safeguards to limit the impact of hot verifier key compromise. Regular verifier set updates now require cold verifier signatures on Spoke chains and, when governance hardening is enabled, on Hub, together with restrictions on authorized submitters. Spoke contracts introduce system candidate lists and enforce mutually exclusive verifier, Finalizer, and Locker roles. Governance messages relayed through the hot verifier path may only disable Finalizers or enable Lockers, while reverse changes require cold-signed authorization. Additional controls include per-token withdrawal limits tracked through hourly buckets, a cold-managed receiver whitelist, restrictions on increasing the Locker pause threshold, and tighter verifier update timelock bounds. The update also introduces a feature-gated pending deposit workflow on Hub. Following deposit processing, an additional callback records the pending deposit. A separate finalizeDeposit entry point verifies hot verifier signatures and the recorded deposit parameters before submitting a Hub finalization command and registering a callback to complete pending deposit processing.

Note this audit only focuses on the codes located in the following directory/file.

- token_bridge.rs
- message_gateway.rs
- bridge_utils.rs
- common.rs
- TokenBridge.sol
- MessageGateway.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

[1] https://github.com/secmgtcoding/dex-bridge.git

Project	Version	Commit Hash
PopDEX Bridge	Version 1	a237a62ad7f0d46ad012cf915fa9179c0df217ce
	Version 2	4b6a3b05c087ccfeb42b15dd6cb240838e72af02

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow

- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **one** potential security issue. Besides, we have **four** recommendations and **two** notes.

- Low Risk: 1
- Recommendation: 4
- Note: 2

ID	Severity	Description	Category	Status
1	Low	Lack of role conflict checks in function `finalizeVerifierUpdate()`	Security Issue	Fixed
2	-	Align custody-chain timing bounds between Hub and Spoke	Recommendation	Fixed
3	-	Add Locker availability checks in function `_setLockerBatch()`	Recommendation	Fixed
4	-	Reject the zero address in system candidate updates	Recommendation	Fixed
5	-	Ensure withdrawals remain counted for the full 24-hour window	Recommendation	Confirmed
6	-	Potential centralization risks	Note	-
7	-	Out of scope dependencies and external logic	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of role conflict checks in function `finalizeVerifierUpdate()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `MessageGateway` on each Spoke chain verifies hot and cold verifier signatures to authorize bridge messages and sensitive governance actions. The contract `TokenBridge` on the same chain assigns the `Finalizer` and `Locker` roles. Accounts with the `Finalizer` role complete withdrawals, while accounts with the `Locker` role vote to pause the bridge and submit authorized recovery operations. To separate these responsibilities, the protocol requires different addresses for verifier membership, the `Finalizer` role, and the `Locker` role.

To enforce this separation, the function `updateVerifiers()` in the contract `MessageGateway` checks proposed hot and cold verifier addresses for existing `Finalizer` or `Locker` roles before queuing an update. The queued update takes effect only after a timelock, so role assignments may change before finalization. However, the functions `_setFinalizerBatch()` and `_setLockerBatch()` in the contract `TokenBridge` check verifier conflicts against only the current verifier set. They do not check proposed addresses in the queued verifier set.

As a result, a proposed verifier address can receive the `Finalizer` or `Locker` role during the timelock. The function `finalizeVerifierUpdate()` in the contract `MessageGateway` then applies the queued verifier set without repeating the role conflict checks. The address therefore retains its `Finalizer` or `Locker` role after joining the verifier set.

```
449 function finalizeVerifierUpdate(
450     uint64 epoch,
451     address[] calldata hotVerifiers,
452     address[] calldata coldVerifiers,
453     uint256[] calldata powers
454 ) external nonReentrant whenNotPaused {
455     PendingVerifierUpdate memory pending = pendingVerifierUpdate;
456     if (pending.paramsHash == bytes32(0)) revert PendingVerifierUpdateNotFound();
457     if (block.timestamp <= pending.executeAfter) revert VerifierUpdateTimelockNotElapsed();
458     bytes32 paramsHash = _verifierParamsHash(epoch, hotVerifiers, coldVerifiers, powers);
459     if (paramsHash != pending.paramsHash || epoch != pending.epoch) revert VerifierSetMismatch();
460     if (epoch <= lastAppliedVerifierEpoch) revert StaleVerifierEpoch();
461     if (hotVerifiers.length != coldVerifiers.length || hotVerifiers.length != powers.length) {
462         revert ArrayLengthMismatch();
463     }
464
465     _validateVerifierSetAndPower(hotVerifiers, coldVerifiers, powers);
466     _clearVerifierSet();
467     _applyVerifierSet(hotVerifiers, coldVerifiers, powers);
468     lastAppliedVerifierEpoch = epoch;
469     delete pendingVerifierUpdate;
470     emit VerifierSetUpdated(epoch, hotVerifiers, coldVerifiers, powers);
471 }
```

Listing 2.1: MessageGateway.sol

Impact An address can join the verifier set while retaining the `Finalizer` or `Locker` role, violating the intended separation of duties.

Suggestion Enforce separation between verifier membership, the `Finalizer` role, and the `Locker` role when a queued verifier set takes effect.

2.2 Recommendation

2.2.1 Align custody-chain timing bounds between Hub and Spoke

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the precompiled contract `TokenBridge` on the Hub chain, the function `handle_set_custody_chain_config()` accepts withdrawal dispute periods and verifier update timelocks as short as 10 seconds. However, the contract `TokenBridge` on a Spoke chain requires a dispute period of at least 300 seconds. The contract `MessageGateway` on the same chain also requires a verifier update timelock of at least 7 days. Values below either minimum cause the entire Spoke configuration update to revert. The precompiled contract `TokenBridge` still retains the accepted configuration, leaving the settings inconsistent between the chains.

```
if dispute_period < MIN_CHALLENGE_PERIOD_SECS {
    return Err(PrecompileError::ExecutionError(
        "disputePeriod must be at least 10 seconds".to_string(),
    ));
}
```

Listing 2.2: token_bridge.rs

```
    if (selector == APPLY_CUSTODY_CHAIN_CONFIG_SELECTOR) {
        (
            uint256 disputePeriod,
            uint256 verifierUpdatePeriod,
            uint256 blockTimeMillis,
            uint256 lockerThreshold,
            uint64 version
        ) = abi.decode(data[4:], (uint256, uint256, uint256, uint256, uint64));
        if (version <= runtimeChainConfig.version) {
            revert StaleCustodyChainConfigVersion();
        }
        _checkDisputePeriod(disputePeriod);
        // The verifierUpdatePeriod range is enforced by MessageGateway.syncVerifierUpdatePeriod;
             on failure the whole call reverts.
        // The locker threshold may only decrease: Spoke cannot verify the Hub really authorized
            it, and raising it would defeat locker circuit-breaking.
        if (lockerThreshold == 0) {
            revert InvalidLockerThreshold();
        }
        if (runtimeChainConfig.version != 0 && lockerThreshold > runtimeChainConfig.
            lockerThreshold) {
            revert LockerThresholdIncreaseNotAllowed();
        }
        runtimeChainConfig = RuntimeChainConfig({
            disputePeriod: disputePeriod,
            blockTimeMillis: blockTimeMillis,
            lockerThreshold: lockerThreshold,
            version: version
        });
        _refreshLocalVotePauseState(lockerThreshold);
        IMessageGateway(messageGateway).syncVerifierUpdatePeriod(verifierUpdatePeriod);
        emit CustodyChainConfigUpdated(disputePeriod, blockTimeMillis, lockerThreshold, version);
        return;
    }
```

Listing 2.3: TokenBridge.sol

```
function _checkDisputePeriod(uint256 disputePeriod) internal view {
    if (disputePeriod < MIN_DISPUTE_PERIOD) revert ChallengePeriodTooShort();
    if (disputePeriod > MAX_DISPUTE_PERIOD) revert DisputePeriodTooLong();
}
```

Listing 2.4: TokenBridge.sol

Suggestion Validate custody-chain timing parameters against the target `Spoke`'s supported bounds before accepting the configuration.

2.2.2 Add Locker availability checks in function `_setLockerBatch()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `TokenBridge` on a Spoke chain, local recovery and role management require a caller with the `Locker` role and sufficient cold verifier signatures. However, the function `_setLockerBatch()` does not ensure that any account retains this role after an update through the function `setFinalizerLockerBatch()`. If an authorized update removes the last role holder while the contract is paused, no account can submit local recovery or restore the role. The paused contract `MessageGateway` also rejects messages from the precompiled contract `TokenBridge` on the Hub chain that grant the `Locker` role. Normal recovery therefore remains unavailable.

```
639     if (selector == APPLY_LOCKER_BATCH_SELECTOR) {
640         (address[] memory lockerAddresses, bool[] memory enabled) = abi.decode(data[4:], (address
                []), bool[]));
641         for (uint256 i = 0; i < enabled.length; ++i) {
642             if (!enabled[i]) revert RelayCannotDisableLocker();
643         }
644         _setLockerBatch(lockerAddresses, enabled);
645         return;
646     }
```

Listing 2.5: TokenBridge.sol

```
822 function _setLockerBatch(address[] memory lockerAddresses, bool[] memory enabled) internal {
823     if (lockerAddresses.length != enabled.length) {
824         revert ArrayLengthMismatch();
825     }
826
827     for (uint256 i = 0; i < lockerAddresses.length; i++) {
828         address locker = lockerAddresses[i];
829         bool isEnabled = enabled[i];
830         if (isEnabled) {
831             _requireSystemCandidate(locker);
832             if (finalizers[locker] || _isCurrentVerifier(locker)) revert RoleConflict(locker);
833         }
834         if (!isEnabled && lockers[locker] && lastPauseVoteRound[locker] == pauseVoteRound) {
835             if (pendingPauseVotes == 0) {
836                 revert InconsistentPauseVoteState();
837             }
838             pendingPauseVotes -= 1;
839             lastPauseVoteRound[locker] = 0;
840             emit EmergencyPauseVoteRevoked(locker);
841         }
842         lockers[locker] = isEnabled;
843         emit LockerUpdated(locker, isEnabled);
```

```
844     }
845 }
```

Listing 2.6: TokenBridge.sol

Suggestion Retain at least one account with the `Locker` role after each membership update.

2.2.3 Reject the zero address in system candidate updates

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `TokenBridge`, the functions `addSystemCandidates()` and `removeSystemCandidates()` update the system candidate list without checking whether the addresses in the parameter accounts are zero. It is recommended to validate each address to prevent potential misoperations.

```
1017
1018 /// @custom:selector 0x16bddc6c
1019 function addSystemCandidates(address[] calldata accounts) external onlyRole(
         SYSTEM_CANDIDATE_ADMIN_ROLE) {
1020     if (accounts.length == 0) revert EmptySystemCandidates();
1021     for (uint256 i = 0; i < accounts.length; ++i) {
1022         if (!_systemCandidates.add(accounts[i])) revert SystemCandidateAlreadyExists(accounts[i
                 ]);
1023     }
1024     emit SystemCandidatesAdded(accounts, msg.sender);
1025 }
1026
1027 /// @custom:selector 0x3ba5bc36
1028 function removeSystemCandidates(address[] calldata accounts) external onlyRole(
         SYSTEM_CANDIDATE_ADMIN_ROLE) {
1029     if (accounts.length == 0) revert EmptySystemCandidates();
1030     for (uint256 i = 0; i < accounts.length; ++i) {
1031         if (!_systemCandidates.remove(accounts[i])) revert NotSystemCandidate(accounts[i]);
1032     }
1033     emit SystemCandidatesRemoved(accounts, msg.sender);
1034 }
```

Listing 2.7: TokenBridge.sol

Suggestion Add a check to ensure that each address in the parameter accounts is not zero.

2.2.4 Ensure withdrawals remain counted for the full 24-hour window

Status Confirmed

Introduced by `Version 1`

Description In the contract `TokenBridge` on a Spoke chain, the function `_finalizeWithdraw()` applies a configured per-token withdrawal limit to receivers outside the whitelist. Withdrawals should count toward this limit for a full 24 hours. However, the hourly accounting excludes an entire bucket once its hour index is 24 hours behind the current hour.

For example, a withdrawal at 00:59:59 on day one stops counting at 00:00:00 on day two, after only 23 hours and one second. Another withdrawal can then use the full limit, allowing the combined amount to exceed the configured limit within 24 hours.

```
708 function _finalizeWithdraw(uint128 withdrawalId) internal {
709     PendingWithdraw memory pending = pendingWithdraws[withdrawalId];
710     if (pending.challengePeriodEndTime == 0) {
711         revert WithdrawNotPending();
712     }
713     if (block.timestamp < pending.challengePeriodEndTime) {
714         revert ChallengePeriodNotEnded();
715     }
716     if (_currentBlockNumber() < pending.minFinalizeBlockNumber) {
717         revert FinalizeBlockNotReached();
718     }
719
720     uint64 tokenId = tokenIdByCustodyTokenAddress[pending.tokenAddress];
721     RuntimeTokenConfig memory tokenConfig = tokenConfigByTokenId[tokenId];
722     if (!tokenConfig.withdrawEnabled) {
723         _clearPendingWithdraw(withdrawalId);
724         _sendWithdrawResult(withdrawalId, BridgeMessageTypes.
              WITHDRAW_RESULT_FAILED_WITHDRAW_DISABLED);
725         emit WithdrawFailed(
726             withdrawalId,
727             pending.sourceMessageHash,
728             pending.tokenAddress,
729             pending.amountCanonical,
730             pending.receiver,
731             "WithdrawDisabled"
732         );
733         return;
734     }
735
736     uint8 destDecimals = _getTokenDecimals(pending.tokenAddress);
737     uint256 amountDestRaw = _canonical18ToDestRaw(pending.amountCanonical, destDecimals);
738     bool bypassDailyLimit = withdrawReceiverWhitelist[pending.receiver];
739     if (!bypassDailyLimit) {
740         uint64 currentHour = _currentWithdrawHour();
741         uint256 used24h = _usedWithdrawAmount24h(tokenId, currentHour);
742         uint256 maxDailyWithdrawRaw = maxDailyWithdrawRawByTokenId[tokenId];
743         if (maxDailyWithdrawRaw != 0 && used24h + amountDestRaw > maxDailyWithdrawRaw) {
744             _clearPendingWithdraw(withdrawalId);
745             _sendWithdrawResult(withdrawalId, BridgeMessageTypes.
                  WITHDRAW_RESULT_FAILED_DAILY_LIMIT_EXCEEDED);
746             emit WithdrawFailed(
747                 withdrawalId,
748                 pending.sourceMessageHash,
749                 pending.tokenAddress,
750                 pending.amountCanonical,
751                 pending.receiver,
752                 "DailyWithdrawLimitExceeded"
753             );
754             return;
```

```
755         }
756     }
757
758     uint256 contractBalance = IERC20(pending.tokenAddress).balanceOf(address(this));
759     if (contractBalance < amountDestRaw) {
760         _clearPendingWithdraw(withdrawalId);
761         _sendWithdrawResult(withdrawalId, BridgeMessageTypes.
                WITHDRAW_RESULT_FAILED_INSUFFICIENT_BALANCE);
762         emit WithdrawFailed(
763             withdrawalId,
764             pending.sourceMessageHash,
765             pending.tokenAddress,
766             pending.amountCanonical,
767             pending.receiver,
768             "InsufficientBalance"
769         );
770         return;
771     }
772
773     // Effects
774     _clearPendingWithdraw(withdrawalId);
775     if (!bypassDailyLimit) {
776         _recordWithdrawAmount24h(tokenId, _currentWithdrawHour(), amountDestRaw);
777     }
778
779     // Interactions
780     IERC20(pending.tokenAddress).safeTransfer(pending.receiver, amountDestRaw);
781     _sendWithdrawResult(withdrawalId, BridgeMessageTypes.WITHDRAW_RESULT_SUCCEEDED);
782
783     emit WithdrawFinalized(
784         withdrawalId, pending.sourceMessageHash, pending.tokenAddress, amountDestRaw, pending.
                receiver
785     );
786 }
```

Listing 2.8: TokenBridge.sol

```
364 function _recordWithdrawAmount24h(uint64 tokenId, uint64 currentHour, uint256 amountRaw)
        internal {
365     uint256 bucketIndex = currentHour % WITHDRAW_HOURLY_BUCKET_COUNT;
366     HourBucket storage bucket = withdrawHourBucketsByTokenId[tokenId][bucketIndex];
367     if (bucket.hourIndex != currentHour) {
368         bucket.hourIndex = currentHour;
369         bucket.amountRaw = 0;
370     }
371     bucket.amountRaw += amountRaw;
372 }
```

Listing 2.9: TokenBridge.sol

Suggestion Ensure each withdrawal remains included in the limit calculation for at least 24 hours.

Feedback from the project The current behavior is acceptable, and no changes are needed.

2.3 Note

2.3.1 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles can conduct sensitive operations, which introduces potential centralization risks. On the Spoke chain, the role `SYSTEM_CANDIDATE_ADMIN_ROLE` of the contract `TokenBridge` can directly modify the system candidate list through the functions `addSystemCandidates()` and `removeSystemCandidates()`, affecting eligibility for role assignments and verifier updates. On the Hub chain, the role `BridgeAdmin` can pause the bridge and modify custody configurations without additional signature or timelock requirements, and these governance operations remain executable while the bridge is paused. Additionally, `Lockers` submit cold-signed operations for role management, withdrawal limits, receiver whitelists, and recovery, while execution requires sufficient cold verifier signatures. The off-chain signature aggregation logic is outside the scope of this audit. If the private keys of these privileged accounts are compromised or unavailable, the protocol's operation and recovery may be affected, subject to the applicable authorization checks.

2.3.2 Out of scope dependencies and external logic

Introduced by Version 1

Description The audited code relies on multiple external functions, modules, and dependencies whose implementations were not provided and are therefore considered out of scope. This includes, but is not limited to, internal business logic functions such as `load_balance()`, `transfer_from_token_bridge()`, `compute_required_verifier_power()`, and `check_role!()`, as well as amount conversion utilities like function `canonical_18_to_raw()`.

In addition, all external crates and runtime components referenced via use statements (e.g., `precompile_interfaces`, `reth_storage_api`, `core_state_provider`) were not reviewed. The protocol also depends on a modified execution environment (reth/revm fork) with custom extensions to the precompile execution model.

Accordingly, all external logic and dependencies invoked by the audited code are assumed to be correctly implemented and secure, and were not assessed as part of this audit.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS