AUDIT RESPONSE DOCUMENT

April 21, 2025

Post-Audit Contract Updates

This document outlines the changes made to the XenBurner smart contracts following the security audit report received on April 19, 2025. It details our response to each recommendation and the implementations that have been completed.

Summary of Changes

The audit identified several areas for improvement, primarily focused on best practices and minor usability enhancements. We have carefully reviewed all recommendations and implemented the necessary changes to address them.

Code-Level Changes

The following code changes were implemented in `contracts/XBurnMinter.sol` based on the audit recommendations

1. Minimum Burn Amount Check

- **Audit Recommendation:** Add a check to prevent users from burning an amount of XEN insufficient to generate any base XBURN (due to integer division).
- **Implementation:** The `burnXEN` function now includes a `require(amount >= BASE_RATIO, "InvalidAmount")` check, ensuring that the input `amount` is at least equal to the current `BASE_RATIO` (1,000,000 XEN).
- **Code Change:**

```
function burnXEN(uint256 amount, uint256 termDays) external nonReentrant {

    require(amount >= BASE_RATIO, "InvalidAmount");  // NEW: Minimum burn amount check

    require(termDays >= 1, "InvalidTerm");

    // ... rest of function remains unchanged

}
```

2. NonReentrant Modifier on initializeLiquidity

- **Audit Recommendation:** Add the `nonReentrant` modifier to the `initializeLiquidity` function for consistency and robustness, even though it is an `onlyOwner` function.
- **Implementation:** The `nonReentrant` modifier has been added to the `initializeLiquidity` function definition.
- **Code Change:**

```
function initializeLiquidity(uint256 xenAmount) external onlyOwner nonReentrant {  // UPDATED: Added nonReentrant

    require(!liquidityInitialized, "AlreadyInitialized");

    // ... rest of function remains unchanged

    liquidityInitialized = true;

}
```

3. Single Initialization Flag

- **Audit Recommendation:** Implement a flag to ensure `initializeLiquidity()` can only be called once.

- **Implementation:** Added a state variable `liquidityInitialized` that is checked at the start of the function and set to true at the end.
- **Code Change:**

```
// State variable added at contract level

bool private liquidityInitialized = false;

function initializeLiquidity(uint256 xenAmount) external onlyOwner nonReentrant {

    require(!liquidityInitialized, "AlreadyInitialized");

    // ... function implementation

    liquidityInitialized = true;  // Prevents future calls

}
```

4. Event Emission for Key Functions

- **Audit Recommendation:** Verify that key functions emit events.
- **Implementation:** Added missing events and standardized event emissions across all key functions.
- **Code Change:**

```
// New events added at contract level

event XENBurned(address indexed user, uint256 xenAmount, uint256 nftId, uint256 rewardAmount, uint256 termDays);

event XBURNClaimed(address indexed user, uint256 nftId, uint256 rewardAmount, bool early);

event SwapExecuted(uint256 xenSwapped, uint256 xburnBurned);

// Emitting events in respective functions

function burnXEN(uint256 amount, uint256 termDays) external nonReentrant {

    // ... existing implementation

    emit XENBurned(msg.sender, amount, tokenId, rewardAmount, termDays);

}
```

```
function claimLockedXBURN(uint256 tokenId) external nonReentrant {
    // ... existing implementation
    emit XBURNClaimed(msg.sender, tokenId, rewardAmount, false);
}
function emergencyEnd(uint256 tokenId) external nonReentrant {
    // ... existing implementation
    emit XBURNClaimed(msg.sender, tokenId, baseReward, true);
}
function swapXenForXburn(uint256 minXburnReceived) external nonReentrant {
    // ... existing implementation
    emit SwapExecuted(xenAmountToSwap, xburnReceived);
}
```

5. Gas Optimizations

- **Audit Recommendation:** Mark constants and one-time set variables as `immutable`, cache frequently used state variables.
- **Implementation:** Changed appropriate variables to `immutable` and optimized storage reads.
- **Code Change:**

```
// State variables changed to immutable
address public immutable xenTokenAddress;
address public immutable uniswapRouterAddress;
uint256 public immutable BASE_RATIO = 1_000_000 * 10**18;  // 1M XEN = 1 XBURN
// Example of cached reads in functions
```

```
function calculateAmplifier() internal view returns (uint256) {
    uint256 deploymentTime = deploymentTimestamp;  // Cache storage read
    uint256 daysSinceLaunch = (block.timestamp - deploymentTime) / 1 days;
    // ... rest of function
}
```

Operational Security Measures

1. Owner Renouncement

- **Audit Recommendation:** Renounce the `Ownable` role on the XenBurner contract and the ownership of the XBurnNFT contract after initial setup.
- **Implementation Plan:** We have added this critical step to our deployment checklist. After initializing liquidity and verifying all parameters are correctly set, we will transfer ownership to a multi-sig timelock contract, then ultimately renounce ownership. This will be performed as part of the mainnet deployment process, not as a code change in the contract itself.
- **Timeline:** Ownership will be renounced within 7 days of confirmed successful mainnet deployment.

2. Liquidity Provider Token Security

- **Audit Recommendation:** Lock or burn Liquidity Provider (LP) tokens to ensure that liquidity cannot be rug-pulled.
- **Implementation Plan:** We will lock 100% of the LP tokens received during the `initializeLiquidity` function call for a minimum of 1 year. We will use a trusted third-party locking service such as Unicrypt or Team.Finance for this purpose, with a public transaction verifiable on-chain.
- **Timeline:** LP tokens will be locked immediately after the initialization of liquidity on mainnet.

3. Swap Trigger Incentive

- **Audit Recommendation:** Incentivize the caller of `swapXenForXburn` with a small reward.
- **Implementation:** We have implemented a 5% caller bounty of the threshold (25M XEN), ensuring that users will trigger the swap as soon as the threshold is hit, keeping the loop autonomous.
- **Code Change:**

```
// Added state variables

uint256 public constant SWAP_THRESHOLD = 500_000_000 * 10**18;  // 500M XEN

uint256 public constant CALLER_REWARD_PERCENTAGE = 5;  // 5% of threshold

function swapXenForXburn(uint256 minXburnReceived) external nonReentrant {

    uint256 xenBalance = xenToken.balanceOf(address(this));

    require(xenBalance >= SWAP_THRESHOLD, "InsufficientXEN");
```

```
    // Calculate and send caller reward
    uint256 callerReward = SWAP_THRESHOLD * CALLER_REWARD_PERCENTAGE / 100;
    xenToken.transfer(msg.sender, callerReward);

    // Swap the remaining amount (minus caller reward)
    uint256 xenAmountToSwap = xenBalance - callerReward;
    // ... rest of swap logic
}
```

Final Token Economic Parameters

After careful consideration and in line with the audit recommendations, we have finalized the following parameters for the mainnet deployment:

Final, Immutable Token-Economy Parameters

Parameter	Final Value	Rationale
Base burn → mint rate	1,000,000 XEN → 1 XBURN	Balances scarcity with accessibility, allowing participation from both small wallets and larger holders
Seed-LP amounts	1,000,000,000,000 XEN + 1,000,000 XBURN	Provides sufficient liquidity depth (~$760) and one million tokens of order-book width
XEN burn split	80% destroyed, 20% saved for swap	Balances immediate burn with funding for future buy-and-burn cycles
Swap-and-burn trigger	500,000,000 XEN accumulated	Large enough for meaningful swaps (~475 XBURN burned each time)
Caller bounty	5% of threshold (25M XEN)	Ensures timely triggering of the swap function
Liquidity-pool lock	100% LP tokens, ≥ 1 year	Demonstrates commitment and prevents liquidity removal

Rationale for 1,000,000:1 Ratio

We have carefully selected the 1,000,000:1 (XEN:XBURN) ratio as our base conversion rate for the following reasons:

1. **Balanced Accessibility and Scarcity**

    - At 100M:1, the entry barrier would be prohibitively high for most users
    - At 100K:1, the system could be easily flooded with excess supply
    - The 1M:1 ratio allows ordinary users to participate while maintaining token scarcity

2. **Liquidity Efficiency**

    - This ratio allows for a deeper and more stable order book
    - Pairing 1M XBURN with 1T XEN provides sufficient trading depth
    - Prevents excessive price volatility on smaller trades

3. **Long-Term Price Stability**

    - The burn ratio serves as a natural price anchor or "hard peg"
    - Creates an equilibrium mechanism: if XBURN trades too high, more burning occurs; if too low, burning slows
    - Maintains a fair and reachable parity threshold

4. **Future Growth Potential**

    - Even with significant price appreciation, the entry cost remains reasonable
    - Allows for sustainable protocol growth without excluding new participants
    - Provides ample room for organic market value discovery

Testing and Verification

All implemented changes have been thoroughly tested to ensure they function as intended:

1. **Unit Tests**

    - Added specific tests for the minimum burn amount requirement
    - Verified the single initialization flag works correctly
    - Confirmed nonReentrant modifier functions properly on all modified functions
    - Tested event emissions for all key functions

2. **Integration Tests**

    - Performed complete burn-lock-claim cycle tests on the updated contract
    - Verified the swap function with caller incentive works as expected
    - Tested interactions between XenBurner and XBurnNFT contracts

3. **Gas Analysis**

    - Measured the impact of gas optimizations (immutable variables, storage caching)
    - Confirmed all functions remain well within block gas limits
    - Verified batch operations perform efficiently

User Experience Improvements

In addition to contract changes, we've made several improvements to the user interface based on the audit recommendations:

1. **Clear Visual Indicators**

    - Added lock duration countdown timers
    - Implemented visual indicators showing maturity status for each NFT
    - Created alerts for when locks are available to claim

2. **Transaction Guidance**

    - Added detailed explanations of the burn-lock-claim process
    - Implemented warnings about emergency end penalties (showing both full and reduced reward amounts)
    - Created confirmations showing the expected outcome of each action

3. **Error Handling**

    - Improved error messages when transactions fail
    - Added predictive checks before submitting transactions
    - Implemented user-friendly explanations of technical errors

Conclusion

We have addressed all primary recommendations from the security audit report. The implemented changes substantially improve both the security and usability of the XenBurner protocol while maintaining its core functionality.

The final mainnet deployment will follow a rigorous process that includes the operational security measures outlined above, particularly the owner renouncement and LP token locking, which are critical to establishing trust and security in the protocol.

We believe these improvements, combined with the carefully balanced tokenomic parameters, position XenBurner for a successful and secure launch on the Ethereum mainnet.

END OF DOCUMENT

This document details the implementation of security recommendations from the April 19, 2025 audit report.