BLOCKAPEX

SMART CONTRACT SECURITY

v 1.0

Date: 01-08-2025

Prepared For: Dark Machine

About BlockApex

Founded in early 2021 BlockApex is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

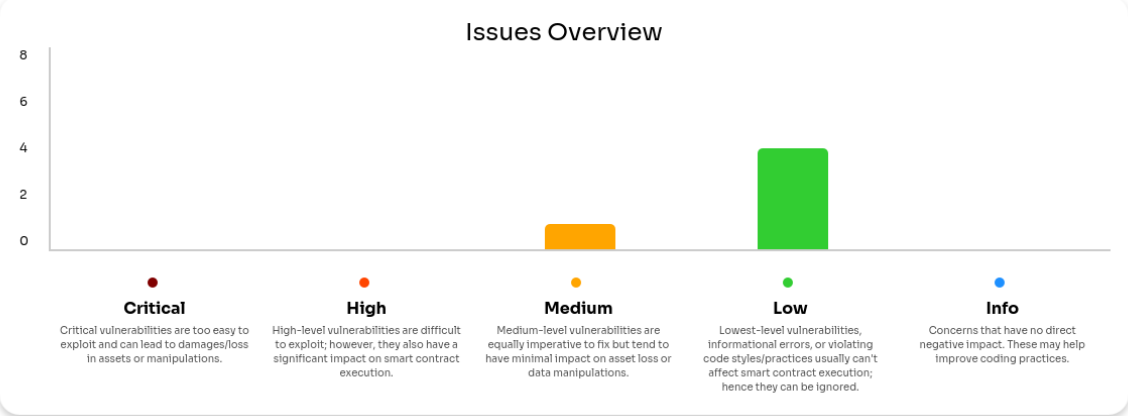
BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

Contents

1 Executive Summary

Our team conducted a Filtered Audit of the Dark Machine smart contracts and dApp. A single auditor independently performed an in-depth manual review and black-box penetration testing to identify potential vulnerabilities and logic flaws. This was complemented by automated tooling to detect common security issues. All findings were manually validated and re-tested to eliminate false positives and confirm their impact.



1.1 Scope

1.1.1 In Scope

Overview of the Contract: Dark Machine is a decentralized staking and reward distribution protocol that enables users to participate in token-based staking programs to earn rewards over time. The platform is designed to offer configurable staking durations, dynamic fee structures, and reward mechanisms that adapt to varying user behaviors. The protocol emphasizes flexible reward distribution and modular fee configurations while maintaining administrative control for governance and economic tuning.

The NewStakingContract is deployed on EVM-compatible blockchains and forms the core of the token staking logic for Dark Machine. It allows users to stake the platform's native token (MXNA), accrue rewards, and claim them based on configurable conditions such as time-locked staking, custom interest rates, and claim/unstake fees. Administrative roles can control parameters such as staking periods, fee structures, and reward logic.

Contract in Scope:

- NewStakingContract.sol — The primary ERC20 staking contract responsible for managing token deposits, reward accrual, claim logic, fee configurations, and administrative control over staking dynamics.

Initial Commit Hash: 6d159a0b0dfdf1de06962ce58ed2f297d0bf314b

Final Commit Hash: a92e9b810d09fd047704367d5284ae3febd6a195

Dapp In Scope:

URL: https://staging–dark-machine-frontend.netlify.app/new-staking

1.1.2 Out of Scope

Any features, contracts, or functionalities not explicitly mentioned in the **In Scope** section are considered out of scope for this audit. This includes any external integrations, off-chain components, or third-party smart contracts interacting with the Dark Machine protocol.

1.2 Methodology

The codebase was audited using a Filtered Audit methodology. A single (1) auditor reviewed the project over approximately 1.5 weeks, beginning with reconnaissance to build context from the code and available documentation and to form testable hypotheses. The audit then progressed to manual code review to surface logic flaws, security vulnerabilities, and opportunities for optimization, alongside evaluations of software/security design patterns, code style, and adherence to best practices. Automated tooling was used to flag common issues, and all findings were manually validated to eliminate false positives and ensure accuracy. In parallel, the dApp underwent black-box penetration testing from an external-attacker perspective to assess exposed functionality and business logic.

1.2.1 Project Overview

The Dark Machine staking protocol is implemented through the NewStakingContract, designed for EVM-compatible blockchains. It enables users to stake MXNA, the protocol's native ERC20 token, and earn rewards based on a time-driven interest model. The contract is highly configurable and upgradeable, utilizing OpenZeppelin's UUPS pattern with role-based access controls for governance and security.

1. **Staking Lifecycle**

Users can stake a minimum amount of 300 MXNA using the stake() function, provided the staking window (defined by stakingStartTime and stakingEndTime) is open. Each stake is recorded in a StakeInfo struct that tracks the staked amount, reward debt, and last update timestamp. The staker is also added to tracking arrays for reward and emergency processing purposes.

2. **Reward Accumulation**

Rewards are calculated based on a default annual interest rate, distributed in 14-day intervals (rewardInterval). The contract supports custom interest periods, allowing protocol managers to set different interest rates for specific durations. When _updateRewards() is called (e.g., during staking, unstaking, or claiming), the contract retroactively calculates and stores pending rewards in rewardDebt, using historical or custom rates depending on the reward timestamp.

3. **Claiming Rewards**

Users can claim their accumulated rewards after a configurable claimStartTime. The reward must be a whole number of MXNA tokens. A claim fee, defined either as a base or time-bound custom rate, is deducted in staking tokens. Notably, fees are rounded up to the nearest whole token unit, increasing effective costs for small claims.

4. **Unstaking Process**

Stakers can partially or fully unstake their MXNA tokens at any time within the allowed staking period. An unstake fee is charged based on either a default or custom fee schedule. If a user's stake drops to zero, their tracking data is removed from the active staker lists.

5. **Emergency Handling**

The contract offers two forms of emergency withdrawal:

- Targeted Emergency Unstake for selected users
- Batched Emergency Unstake for all users in groups (to avoid gas limit issues)

No fee is applied during emergency withdrawals, and rewards are updated before funds are returned.

6. **Administrative Control**

The protocol's design gives elevated control to addresses holding the DEFAULT_ADMIN_ROLE and STAKING_MANAGER_ROLE. These roles can:

- Set or update staking periods, reward intervals, interest rates, and fee parameters
- Change the reward token
- Trigger emergency withdrawal functions
- Upgrade the contract via the UUPS proxy mechanism

This structure allows for high flexibility but also centralizes significant power in a small set of roles.

7. **Upgradability & Modularity**

The NewStakingContract inherits from OpenZeppelin's UUPSUpgradeable, allowing for controlled logic upgrades. A reserved storage gap ensures layout compatibility across future contract versions.

1.3 Questions for Security Assessments

These are the main questions for security assessment, but the evaluation is not limited to these alone.

1. How does the protocol ensure that reward distribution remains efficient and scalable as the number of users increases, without risking transaction failure or excessive gas usage?

2. What measures are in place to restrict access to sensitive administrative functions and prevent unauthorized changes to critical staking configurations?

3. How does the contract validate staking and unstaking actions to ensure that users meet all necessary requirements before token transfers are executed?

4. Are the appropriate protections implemented to prevent reentrancy attacks in functions that involve token transfers, configuration updates, or external interactions?

5. What safeguards exist to ensure that reward tokens are properly managed and cannot be changed or withdrawn in a way that disrupts reward distribution?

6. How does the protocol prevent the misuse of time-based configuration functions to create overlapping or conflicting fee and interest periods?

7. Are there mechanisms to ensure users cannot accumulate rewards beyond the intended staking period, and that reward accrual aligns with the defined staking lifecycle?

8. Does the protocol enforce clear boundaries on adjustable parameters such as fees and thresholds to prevent arbitrary or harmful configurations?

9. How does the contract maintain the separation of protocol-owned funds from user deposits to avoid accidental or unauthorized withdrawals?

10. Are there fallback mechanisms or validations in place to ensure consistent protocol behavior in edge cases such as zero-amount claims, expired configurations, or unexpected admin actions?

1.4 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.5 Summary of Findings Identified

S.NO	SEVERITY	FINDINGS	STATUS
#1	MEDIUM	Missing DMARC Record Allows Email Spoofing and Phishing Attacks	FIXED
#2	LOW	Missing Overlap Validation in Update Functions for Custom Periods	FIXED
#3	LOW	updateTotalRewardDebt Function Becomes Unusable at Scale Due to Gas Limits	ACKNOWLEDGED
#4	LOW	Rounding Up Fee to Full Ether is Unfair	FIXED
#5	LOW	Missing Security Headers Expose Users to Client-Side Attacks	FIXED

2 Findings and Risk Analysis

2.1 Missing DMARC Record Allows Email Spoofing and Phishing Attacks

Severity: Medium

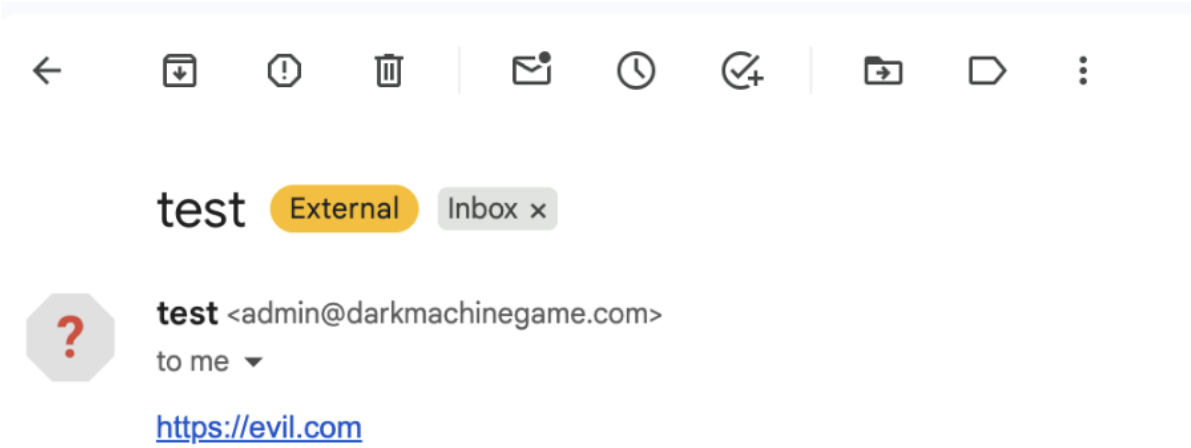
Status: Fixed

Location:

1. https://darkmachinegame.com/

Description: The domain under review does not enforce a DMARC (Domain-based Message Authentication, Reporting, and Conformance) policy. This missing security control allows malicious actors to spoof emails from the domain, which can be exploited for phishing attacks and social engineering campaigns targeting users, partners, or internal teams.

DMARC builds on SPF and DKIM records to prevent unauthorized senders from forging the domain in the “From” header of emails. Without it, attackers can impersonate official communication channels—posing as the platform’s support team, security team, or even executives—to manipulate victims into sharing sensitive information or authorizing fraudulent actions.



Proof of Concept: To validate this issue, a spoofed email was sent using a fake mailer:

1. Set the “From” address to a legitimate-looking email (e.g., admin@darkmachinegame.com)
2. Sent the spoofed email to a test inbox.
3. The email bypassed common spam filters and landed in the inbox, appearing as if it originated from the official domain.

4. DMARC validation headers were missing, confirming that the domain does not enforce a DMARC policy.

Recommendation: Implement a strict DMARC policy for the domain as follows:

1. Add the following TXT record to your domain's DNS:

```
Host: _dmarc.example.com
Type: TXT
Value: v=DMARC1; p=reject; rua=mailto:dmarc-reports@example.com
```

2. Monitor DMARC reports for visibility into unauthorized email attempts.
3. Ensure that SPF and DKIM are properly configured to align with the DMARC policy.
4. Gradually move from p=none → p=quarantine → p=reject as confidence improves.

This will prevent external actors from using your domain in forged email headers, reducing phishing risk and improving overall email deliverability and trust.

This will prevent external actors from using your domain in forged email headers, reducing phishing risk and improving overall email deliverability and trust.

2.2 Missing Overlap Validation in Update Functions for Custom Periods

Severity: Low

Status: Fixed

Location:

contracts/NewStakingContract.sol

Description: The contract provides admin-controlled mechanisms to define and manage custom time-based configurations for interest rates, unstake fees, and claim fees using the following pairs of functions:

- **addCustomInterestPeriod()**
- **addCustomUnstakeFeePeriod()**
- **addCustomClaimFeePeriod()**

Each of these add functions correctly includes validations to prevent overlap with existing periods, ensuring clear and non-conflicting schedules.

However, the corresponding update functions:

- **updateCustomInterestPeriod()**
- **updateCustomFeePeriod()**
- **updateCustomClaimFeePeriod()**

do not perform any overlap validation, allowing the admin to modify an existing period into a time range that overlaps with others. This breaks the consistency enforced by the add functions.

Code Affected

```
function updateCustomClaimFeePeriod(
        uint256 index,
        uint256 startTime,
        uint256 endTime,
        uint256 feeRate
    ) external onlyRole(STAKING_MANAGER_ROLE) {
        require(index &lt; customClaimFeePeriods.length, &#x27;Invalid index&#x27;);
        require(startTime &lt; endTime, &#x27;Start time must be before end time&#x27;);
        require(feeRate &gt;= 0, &#x27;Fee rate must be greater than zero&#x27;);

        customClaimFeePeriods[index] = CustomFeePeriod({startTime: startTime, endTime:
            endTime, feeRate: feeRate});

        emit CustomClaimFeePeriodAdded(startTime, endTime, feeRate);
    }

//Note: All functions mentioned above contain the same update logic.
```

Recommendation: Ensure that the updated ranges do not overlap with any other existing periods (excluding itself). Maintains consistent and non-conflicting boundaries for all custom configurations.

2.3 updateTotalRewardDebt Function Becomes Unusable at Scale Due to Gas Limits

Severity: Low

Status: Acknowledged

Location:

contracts/NewStakingContract.sol

Description: The **updateTotalRewardDebt()** function is designed to recalculate the totalRewardDebt across all stakers by iterating through the entire allStakers array. While this helps maintain accounting accuracy, the function reads and aggregates user-level rewardDebt values in a single loop without batching. As the number of stakers grows (e.g., beyond ~400-500 depending on gas usage and chain limits), the call may exceed block gas limits and fail, making this function unusable.

Code Affected

```
function updateTotalRewardDebt() external onlyRole(STAKING_MANAGER_ROLE) {
    uint256 newTotalRewardDebt = 0;

    for (uint256 i = 0; i &lt; allStakers.length; i++) {
        address user = allStakers[i];
        StakeInfo storage stakeInfo = stakes[user];
        newTotalRewardDebt += stakeInfo.rewardDebt;
    }

    totalRewardDebt = newTotalRewardDebt;
    emit TotalRewardDebtUpdated(newTotalRewardDebt);
}
```

Recommendation: Implement a batched version of the function, similar to getTotalRewardDebtWithPendingBatched(), allowing partial updates across multiple calls.

2.4 Rounding Up Fee to Full Ether is Unfair

Severity: Low

Status: Fixed

Location:

contracts/NewStakingContract.sol

Description: The contract implements a rounding mechanism for both unstake and claim fees that rounds the fee up to the nearest full ether whenever there is any fractional remainder. This logic is present in both **calculateUnstakeFee()** and the **claim()**

Code Affected

```
if (fee % 1 ether != 0) {
    fee = ((fee / 1 ether) + 1) * 1 ether;
}
```

Recommendation: Avoid rounding up to a full ether. Instead:

- Use fixed-point precision (e.g., round to 1e12 if needed for dust protection).
- Or retain the fee as-is, since ERC20 tokens (like MXNA) support fractional units by design.
- Transparently emit the calculated fee value in an event so it can be audited externally.

2.5 Missing Security Headers Expose Users to Client-Side Attacks

Severity: Low

Status: Fixed

Location:

1. https://staging–dark-machine-frontend.netlify.app/new-staking

Description: The web application responds with an incomplete set of HTTP security headers. While it correctly implements Strict-Transport-Security and X-Content-Type-Options, several important security headers are missing, which could expose users to a variety of client-side attacks, including cross-site scripting (XSS), clickjacking, referrer leakage, and unauthorized access to browser features. These headers are widely recommended as part of secure application deployment practices and are recognized in major security benchmarks (OWASP, Mozilla Observatory, Google Web Fundamentals).

The following headers were missing in the HTTP response:

1. **Content-Security-Policy** Without it, attackers can inject malicious scripts and execute XSS attacks.
2. **X-Frame-Options** The site can be embedded in an iframe, making it vulnerable to clickjacking.
3. **Referrer-Policy:** Sensitive URLs can be leaked to third-party sites via the Referer header.
4. **Permissions-Policy** Browsers may expose sensitive features (camera, mic, geolocation) by default
5. **Cross-Origin-Embedder-Policy** No protection against speculative side-channel attacks in modern browsers.
6. **Cross-Origin-Opener-Policy** Allows shared browsing contexts, increasing XSS and data leak risks.
7. **Cross-Origin-Resource-Policy** Allows external sites to fetch protected assets.
8. **X-Permitted-Cross-Domain-Policies** May allow Flash/Java applets to access sensitive cross-domain data
9. **Clear-Site-Data:** No control over clearing cookies/storage upon logout or sensitive transitions.

Additionally, the response includes:

X-Powered-By: Next.js

This discloses backend technology to attackers and should be removed to reduce fingerprinting risk.

Recommendation: * Implement the above mentioned HTTP headers with secure policies * Also, remove or suppress the X-Powered-By header to avoid backend fingerprinting

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts