Multiplier.Finance MCL Threat Model

Multiplier.Finance

25 May 2021
Version: 1.0

Presented by:
Kudelski Security Research Team
Kudelski Security – Nagravision SA

Corporate Headquarters
Kudelski Security – Nagravision SA
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

DOCUMENT PROPERTIES

Version:	1.0
File Name:	0525 Findings_Multiplier_Finance_Final_Report Threat Model
Publication Date:	25 May 2021
Confidentiality Level:	For Public Disclosure
Document Owner:	Scott Carlson
Document Recipient:	Multiplier Finance Project Team
Document Status:	Approved

SYSTEM & APPLICATION THREAT MODEL REPORT

INTRODUCTION

Methodology

This document is the result of analysis from the Kudelski Security and the Global Assessment team. The team reviewed documentation provided by Multiplier.Finance and conducted three workshops/meetings specific to prioritized components within the suite of applications known as the “multiplier.finance” web system. Each component in what we identified as the business process flow was reviewed against a list of functional requirements listed in the provided Litepaper, industry best practice, and modern attack vectors to identify potential abuse cases against the platform by specific threat actors defined during the engagement. A slightly modified STRIDE Threat Modeling approach was leveraged to determine threats across the platform.

Kudelski Security maintained a complete and consistent view across the known components and followed a systematic approach. First threat actors of concern were identified and data flows between the system components were requested. Based upon the understanding of each component from documentation and the interviews, remote follow-up meetings were held with team members of Multiplier.Finance for clarification of any technical or functional details. The STRIDE category table below was used to provide a high level overview of potential threats identified through this process. Upon completion of the STRIDE table threats were enumerated with consideration of threat actors, trust zones, and scoped components to provide a thorough review of each potential threat.

Observations noted in this threat model are not indicators of vulnerabilities within the multiplier.finance web system. The observations noted here were used to drive investigation in the subsequent code review. Related vulnerabilities and findings are noted in the references section of each observation.

This document was provided for peer review to the code review and security assessment team to be used in their technical evaluation of the product upon completion of the model.

Components

The application architecture is split into three major components of interest across the model. We have noted supporting components where relevant such as databases, third party services, or supporting platforms. Logical applications components were also included by function to the platform considering the financial and transactional nature of the application. The components were assessed at the application and functional level in order to consolidate network, blockchain, and logic level threats into single entities and to appropriately target and isolate the scope of the model.

COMPONENT	DESCRIPTION
Frontend	Typescript application that exposes the UI
MCL-SmartContract	Contact code and logic deployed on the Binance Smart Chain
Postgres Database	Repository for transactions for view only since graphing is not available
ECS Registry	Container repository for built front end code
ECS Container	Platform on which the front-end component is deployed
Wallet	Wallet providers integrated with the platform
Lending Function	Enables users to stake assets for collateral or liquidity for other users
Liquidation Function	Enables users to participate in liquidation of under collateralized assets
Borrowing Function	Enables users to borrow for collateral
Voting Function	Enables users that have earned tokens to participate in voting
Pausing Function	Emergency function to pause operations
Frontend	Typescript application that exposes the UI
MCL-SmartContract	Contact code and logic deployed on the Binance Smart Chain
Postgres Database	Repository for transactions for view only since graphing is not available
ECS Registry	Container repository for built front end code
ECS Container	Platform on which the front-end component is deployed

Assets

ASSET	DESCRIPTION
Front End Code	The deployed front end code prior to being deployed to the chain
Logging Database	The data within the postgresql RDS instance used to present information to end users
Private Keys	Private keys for the purposes of conducting on-chain transactions using the MCL wallet(s)
MXX and bMXX Tokens	Access and possession of tokens
Contract Owner	Although not a physical asset, contract ownership of the instantiated contract is noted here for the purposes of tracking and threats associated with unauthorized use of administrative functions
Contract	The instantiated contract on chain

Trust Zones

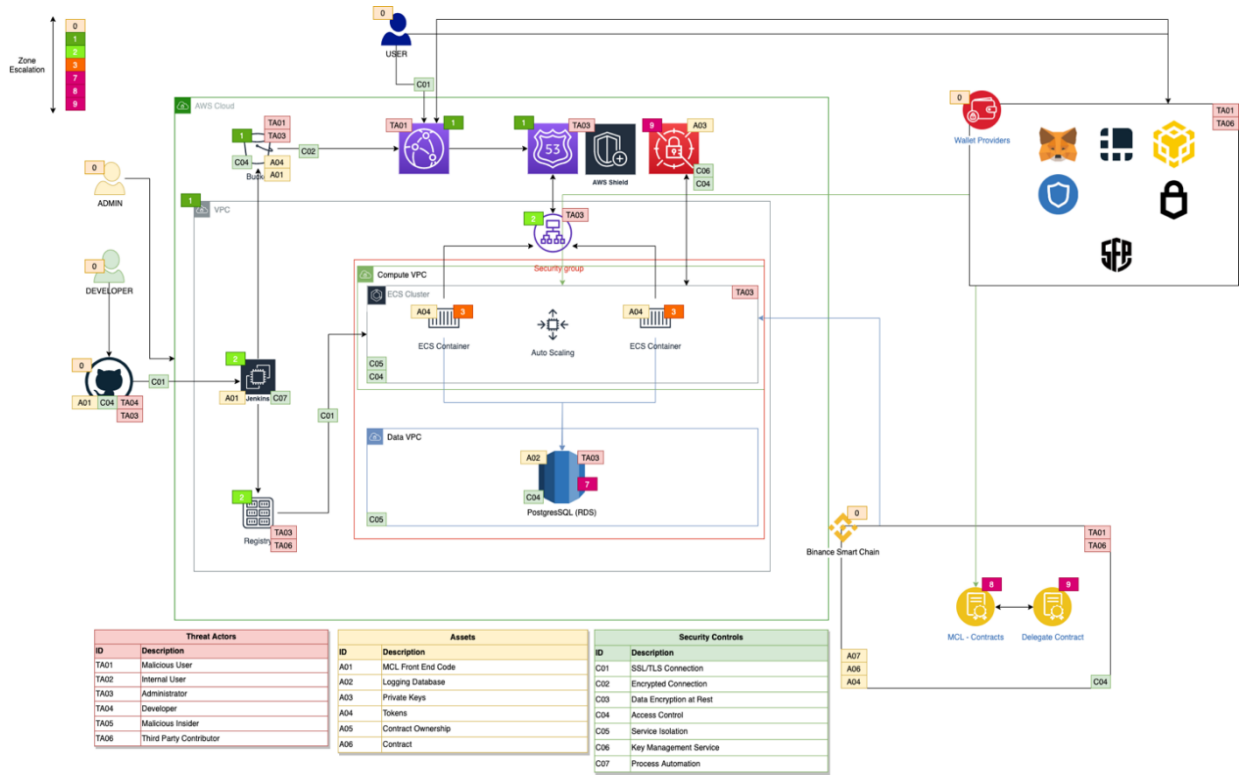
ZONE	DESCRIPTION	INCLUDED COMPONENTS
Internet	The externally facing, wider internet zone	Wallet, Frontend
Binance Smart Chain	The externally facing blockchain	MCL Contracts, Wallet
Compute Subnet	Internal zone holding compute resources for front end interface	ECS Containers, Front End Application
Data Subnet	Internal zone holding the graphing data only	Postgres Database
Client Browser	External zone isolated to each user	Wallet Connection

Threat Actors

It should be noted that threat actors can be individuals or entities that do not intend to compromise or exploit the system intentionally. These actors are considered to account for insider threat, stolen credentials, or exploit via proxy.

ACTOR	DESCRIPTION
External User	Any entity accessing the front end components or contracts directly
Internal User	Any non-administrative user that serves as a representative of MCL
Malicious Actor	Any user external or internal whose primary intent is associated with malicious behavior or nefarious activity
Developer	An internal non administrative user with access to source code for either the front end applications or smart contracts
Administrator	An internal user with administrative access to the supporting systems, smart contracts, or source code
Third Party Contributor	An entity responsible for delivering third party code or resources into the system that is outside the realm of control for MCL

DIAGRAMS



Threat Actors	
ID	Description
TA01	Malicious User
TA02	Internal User
TA03	Administrator
TA04	Developer
TA05	Malicious Insider
TA06	Third Party Contributor

Assets	
ID	Description
A01	MCL Front End Code
A02	Logging Database
A03	Private Keys
A04	Tokens
A05	Contract Ownership
A06	Contract

Security Controls	
ID	Description
C01	SSL/TLS Connection
C02	Encrypted Connection
C03	Data Encryption at Rest
C04	Access Control
C05	Service Isolation
C06	Key Management Service
C07	Process Automation

THREATS, ABUSE, AND OBSERVATIONS

Observation KSI-001: Using components with known vulnerabilities

Severity Potential (Critical):

CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Threat: A contract uses an older version of Solidity, OpenZeppelin, or other libraries with known flaws, cryptographic or otherwise that allows for unexpected behavior in the contract.

Description: Multiple flaws have been discovered in Solidity versions over the years and if these are not remedied the functionality can be used to transfer funds or exploit the money market system.

Recommendations: Ensure that Solidity is pinned to a current version and that external libraries are kept up to date.

Findings: Normally, an older version of a dependent library is considered a security risk, because the dependent system of Aave, OpenZeppelin also depends on this external library, this risk is not applicable and does not confer risk.

Reference: KS-ML-F-01

Observation KSI-002: Lack of input validation that results in a loss of funds

Severity Potential: (Critical)
CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:H/A:L

Threat: A user inputs invalid data to a contract that is not validated appropriately resulting in a loss of funds or transfer to incorrect address.

Description: Numerous scenarios exist where input needs to be validated as expected. In an exchange where a variety of tokens and currencies are exchanged it is important to determine whether the transaction is appropriate. Otherwise users may transfer tokens to the wrong token type address or currency and lose the funds entirely. Additionally, if addresses are not validated they can be subject to padding attacks or code execution.

Recommendations: It is important to understand the inputs coming into each contract and to validate them accordingly. This is especially relevant for arbitrary values in strings. This can also be covered by writing unit tests to simulate manipulated values to ensure error handling and fallback functions behave accordingly.

Findings: No findings based on this observation were found during the code review.

Observation KSI-003: Market manipulation through collusion or market flooding

Severity Potential: (Critical)

CVSS v.3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:H

Threat: A user is able to flood the market with funds in order to drive the interest rate down in order to provide a more favorable interest rate for borrowing or alternatively borrow a large amount in order to drive the incentive upwards.

Description: With a low amount of funds there is an increase in the possibility that the market is subject to whale collusion or a single entity driving interest rates up or down through a large deposit or withdrawal from the lending pool.

Recommendations: Ensure that there are sufficient protections in place for large deposits or withdrawal from the market. This could be a hard limit or solved through active monitoring of the blockchain.

Findings: No findings based on this observation were found during the code review.

Observation KSI-004: Direct execution of delegate contracts in multi-phase process

Severity Potential: (Critical)

CVSS v.3.1 Vector: AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:H

Threat: A user is able make direct calls into smart contracts intended for multi-step or delegate processing which allows the user to execute transfers or contract functions prior to authorization by a previous step.

Description: Direct execution of contracts on chain is not necessarily restricted on the chain and a failure to properly set up administrative functions within the contract could result in functionality being executed in a way that isn't intended. This could be used for the transfer of funds, changes of address, or destruction of the contract.

Recommendations: Ensure that access controls specify the preceding contract address if the intention is that transactions happen in a specific sequence. Alternatively, you can take steps to ensure that all operations happen in a single transaction.

Findings: No findings based on this observation were found during the code review.

Observation KSI-005: Replay of valid transaction

Severity Potential: (High)

CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:H/A:L

Threat: The executor of the contract is able to reuse transactions to obtain additional bMXX Tokens by replaying the transaction before validating functions are written.

Description: There are several conditions in which a malicious user can replay a signed transaction in an attempt to exploit fund transfer by withdrawing multiple times perhaps prior to a balance being updated.

Recommendations: Ensure that all functions are cleaned to ensure that a valid transaction is accomplished in one transaction set.

Findings: No findings based on this observation were found during the code review.

Observation KSI-006: Weakness to front running attacks

Severity Potential: (High)

CVSS v3.1 Vector: AV:A/AC:H/PR:H/UI:R/S:C/C:N/I:H/A:H

Threat: Since all transactions are visible for a short while before being executed, bots and observers of the network can see and react to transactions before they are written to a block.

Description: Front running is specific to transactions but results when a contract is seen, copied, and executed before the original sender can get the transaction written. This is typically conducted by miners who are able to see an advantageous contract and send their own using themselves as the msg.sender. This is essentially stealing a transaction for the benefit of the attacker.

Recommendations: Front running is difficult to defend against, but one solution is to set the minimum and maximum price range to limit price slippage. Additionally, in some cases commit and reveal schemes can be used to limit visibility of the transaction.

Findings: No findings based on this observation were found during the code review.

Observation KSI-007: The use of inherently weak randomness to generate confidential values

Severity Potential: (High)
CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:H/A:H

Threat: The executor of the contract is able to predict the generated address or information within a contract that uses random values.

Description: Randomness is difficult on the blockchain. If a malicious actor can predict the destination address, the address specific deployment, or another confidential value, they may be able to compromise the deployment of the contracts.

Recommendations: Do not use random numbers to generate values intended to be private. While values can be hard to predict, they are not perfect and malicious users can generally replicate it and attack the function relying on it. Consider random values public and adjust logic accordingly.

Findings: No findings based on this observation were found during the code review.

Observation KSI-008: Owner tampering and manipulation

Severity Potential: (High)

CVSS v.3.1 Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N

Threat: Contract initializer is able to change the owner of the contract.

Description: A common logic flaw is a failure to properly initialize the owner of the contract. If the owner can be changed after initialization, it may allow the caller to drain the contract of its funds or execute administrative functions.

Recommendations: Ensure that the owner cannot be arbitrarily changed through public functions and that appropriate access control is in place for internal and private functions.

Findings: Initially, there was a flaw which could lead to an extremely high interest rate, which was resolved by the project team to our satisfaction, fixing the interest rate maximum at 10%.

Reference: KS-ML-F-02

Observation KSI-009: Misuse of Solidity functions for determining contract ownership

Severity Potential: (High)
CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:L

Threat: A contract instance is able to execute the contract as the contract owner due to a misuse or a misunderstanding of the underlying code.

Description: Some functions such as tx.origin will always point to the contract owner. If this is used during the contract instantiation it is possible that methods executed by the instance are executed as an administrator of the contract if tx.origin is used for authorization.

Recommendations: Ensure that logic is suitable for each inherent function. In the case of tx.origin as authorization, msg.sender will check the executor of the contract rather than the owner.

Findings: No findings based on this observation were found during the code review.

Observation KSI-010: The ECS registry is compromised

Severity Potential: (Medium)

CVSS v3.1 Vector: AV:L/AC:H/PR:H/UI:N/S:C/C:L/I:H/A:LA

Threat: The ECS registry for application containers or supporting containers is loaded with malicious containers or incorrect containers.

Description: An administrator or a malicious user with access to the ECS registry can manipulate containers within the registry. If these are not validated, they can be used to generate unexpected or malicious behavior within the application.

Recommendations: Follow AWS best practices for securing ECR including developing a tag based access control to provide fine-grained access, adjusting IAM policies, enabling encryption, and leveraging AWS PrivateLink for back end access.

Findings: No findings based on this observation were found during the code review.

Observation KSI-011: DNS Hijacking

Severity Potential: (Medium)

CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L

Threat: The MCL domain is trusted by the wallet extension and later spoofed on the user's network or through a DNS hijacking attack.

Description: The wallet trust is established by the domain and it is unclear what controls are applied to each supported wallet to ensure the validity of the domain and underlying IP address and/or load balancer. This could result in a trust between the user and the domain name allowing for a malicious actor to leverage the trust between the wallet and MCL to redirect funds. While most of the defense is on the user there are some techniques that will address this server-side.

Recommendations: Configure DNSSEC for Route53 so that the DNS resolver establishes a chain of trust for responses from intermediate resolvers, so that an error will be returned if a user is subject to a man-in-the-middle or DNS hijacking attack.

Findings: No findings based on this observation were found during the code review.

Observation KSI-012: Exposure of administrative functions in nested access control

Severity Potential: (Medium)
CVSS v3.1 Vector: AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:L

Threat: During deployment of the MCL Smart contract the owner is established as a non-MCL entity.

Description: If the contract owner is changed in any delegate contract this could expose administrative functions within each contract within the MCL eco system. If the contract owner is changed and deployed to the chain there is not much that can be done to revoke that trust. This could provide the owner of delegate contracts access to those private and internal methods if contract to contract access control is not sufficient. Depending on the contracts deployed this could result in funds being exchanged or drained from MCL wallets.

Recommendations: Review all access control of external functions and modifiers to ensure the chain of access is not inadvertently broken. This is especially relevant when source code is available to create malicious contracts that may be able to pass simple access controls that are not based on a specific address.

Findings: No findings based on this observation were found during the code review.

Observation KSI-013: Address tampering in official channels and administrative contracts

Severity Potential: (Medium)
CVSS v3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:L

Threat: A destination or source wallet addressed is changed in the application and honored by the smart contract.

Description: The user has access to the front end application and in some cases can execute the smart contracts arbitrarily. If this data (including addresses) is manipulated off the chain, in the browser, the downstream consequences to the execution of that contract could be catastrophic. This includes man-in-the-middle, man-in-the-browser, or simply direct contract execution with arbitrary data.

Recommendations: Ensure that all address changing functions are marked as administrative through appropriate access control such as multi-sig and access modifiers. If addresses are not constants and can be changed it is essential that they are considered a primary asset.

Findings: No findings based on this observation were found during the code review.

Observation KSI-014: Acquisition of tokens prior to launch

Severity Potential: (Medium)

CVSS v.3.1 Vector: AV:L/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:N

Threat: A user is able to obtain bMXX tokens before launch and hold them prior to release and stack votes within the system, potentially obtaining a majority.

Description: Since the bMXX tokens are used to govern the overall platform a compromise of the tokens could steer the direction of the platform overall. This could be an issue if there is a weakness in the initial contract or if there is a function based on time derived from block headers.

Recommendations: Ensure that the bMXX contracts do not rely solely on time to determine the distribution of tokens. It is also essential that access controls are restricted to administrators.

Findings: No findings based on this observation were found during the code review.

Observation KSI-015: Unauthorized execution of the pause and unpause functions

Severity Potential: (Medium)

CVSS v3.1 Vector: AV:N/AC:L/PR:H/UI:N/S:C/C:N/I:N/A:H

Threat: The pause function is executed by a malicious entity or due to a lack of appropriate permissions within the contract.

Description: The pause function is an administrative function that allows a user to pause or continue transactions if an investigation into activities or a halting of the platform is required. Failure to properly authorize this function could result in a malicious actor continuing to abuse the system when a flaw is discovered or cause a denial of service

Recommendations: Always review the pause function for appropriate assignment of the pauser role and execution of the pause functions. It may be beneficial to consider using internal functions for this purpose.

Findings: No findings based on this observation were found during the code review.

Observation KSI-016: Unauthorized self-destruction of contract

Severity Potential: (Medium)

CVSS v3.1 Vector: AV:N/AC:L/PR:H/UI:N/S:C/C:N/I:N/A:H

Threat: A malicious entity is able to self-destruct a contract without authorization or intention by MCL.

Description: The ability to self-destruct contracts should be restricted to the owner of the contract and in cases where this is not properly coded an anonymous malicious entity may be able to self-destruct a contract and transfer unused funds or cause a denial of service.

Recommendations: The self-destruct function is an administrative function that allows a user to destroy the contract. Always review for appropriate contract ownership assignment and access modifiers.

Findings: No findings based on this observation were found during the code review.

Observation KSI-017: Tampering of interest rates and other funding modifiers

Severity Potential: (Medium)

CVSS v3.1 Vector: AV:N/AC:L/PR:H/UI:N/S:C/C:N/I:H/A:H

Threat: An end-user or malicious entity is able to manipulate or change interest rates or modifiers within the contract or as trusted by the client application in order to present themselves a more advantageous rate or additional funds.

Description: The interest rates and modifiers to transactions should all be held within the contract and not necessarily trusted as an input from the end user otherwise this data could be tampered with in flight or by the client application itself. Additionally, if users are able to input these rates directly into the contract or its delegates the contract will execute accordingly.

Recommendations: If the change of interest rates is exposed as a function, ensure that these functions' access modifiers are appropriately configured. Additionally, ensure contract ownership is appropriately defined and cannot be modified by an arbitrary user.

Findings: The system initially displayed a higher value than was possible to actually borrow, potentially resulting in user misunderstanding even though the system would not let the action complete. This graphical error was resolved by the development team to display accurate values appropriately.

Reference: KS-MCL-F-18

Observation KSI-018: Arithmetic flaws result in logic checking flaws allowing for the unauthorized withdrawal of funds

Severity Potential: (Medium)

CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:H

Threat: Logic flaws enable a user or address to borrow without providing sufficient collateral.

Description: If an end user or malicious entity is able to exploit a logic flaw so that they do not provide sufficient collateral to the lending pool, this could result in an immediately unhealthy account and result in negative effects to rates or unauthorized transfer of funds.

Recommendations: Review the arithmetic associated with approving a user to borrow funds and design unit tests that test for input of negative and invalid values to ensure appropriate error handling and fallback.

Findings: There are No applicable findings - The arithmetic operations in the system are based on an 18 digit token length, therefore the rounding errors are not applicable as they apply to numbers of 24 digits in length.

Reference: KS-MCL-F-03

Observation KSI-019: Use of assembly functions enables arbitrary code execution

Severity Potential: (Medium)

CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:H/A:H

Threat: The sender to the contract is able to insert arbitrary code due to some use of assembly and inclusion of user supplied values that read the assembly code.

Description: The use of assembly code is permitted within contracts but is not recommended. If user input is passed to functions that allow this it could result in the execution of arbitrary code.

Recommendations: Avoid the use of assembly in all contracts and rely on built-in functions or abstracted libraries to achieve the result if possible.

Findings: No findings based on this observation were found during the code review.

Observation KSI-020: Denial of service due to the failure, destruction, or compromise of an underlying support component

Severity Potential (Medium):

CVSS v3.1 Vector: AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H

Threat: The RDS Postgres server, Jenkins server, S3 bucket, Cloudfront, or Route53 go offline or are rendered unusable.

Description: The supporting system may not be required to execute contracts but it is necessary for end-user access to funds and contracts. If any of the supporting components go down the system is rendered useless and users cannot access their funds, tokens, or liquidity.

Recommendations: It is noted that while there are certain elements of redundancy and resilience inherent to cloud service providers like AWS, there are elements of the architecture that are not resistant to failure. This is relevant especially to down time in regional services such as ECS, ECR, and RDS.

Findings: No findings based on this observation were found during the code review.

Observation KSI-021: Compromise due to leaked administrative credentials

Severity Potential (Medium):

CVSS v3.1 Vector: AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H

Threat: Administrative accounts to underlying infrastructure are compromised due to insider threat, stolen, or compromised credentials.

Description: Any underlying support system has access to proprietary and private information, especially with regard to cloud services such as secrets management, IAM, and storage. If these credentials are compromised it could result in loss of complete control in the system and the contract management.

Recommendations: Follow best practices for all infrastructure accounts such as in AWS. This includes the avoidance of root account usage, MFA, strong passwords, and the use of programmatic access for changes to infrastructure.

Findings: No findings based on this observation were found during the code review.

Observation KSI-022: The client application is cloned or spoofed

Severity Potential: (Low)

CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:R/S:C/C:N/I:L/A:N

Threat: A malicious user is able to clone or modify the MCL lending consumer application and present it as a legitimate entity that makes transactions on chain for the purposes of hijacking transactions or redirecting intended funds to a different destination.

Description: The source code for the MCL site is primarily available from the site itself and subject to being cloned and independently re-used. Since the application makes direct transactions to the chain it is possible that a malicious entity spoofs this site and points to other destination wallets or funding resources. This is potentially damaging to the brand and to the consumers. Additionally, if the site is modified in some way due to lack of proper authentication/authorization to the infrastructure and presented as the legitimate site any malicious transactions would appear legitimate to the user.

Recommendations: Restrict access to the client web application code as much as possible. By its nature, a javascript application it is nearly impossible to prevent cloning of the site itself, but steps can be taken to reduce the risk by ensuring that the source code is not readily available. Additionally, ensure that the application is configured in a way that prevents any private information from being hard coded into the source. Additionally, ensure that communication is clear and readily available to your user base that details the appropriate locations for the application such as domain and URL.

Findings: No findings based on this observation were found during the code review.

APPENDIX D: CONTACTS

NAME	POSITION	CONTACT INFORMATION
Scott Carlson	Director – Security Architecture	scott.carlson@kudelskisecurity.com
Ryan Spanier	VP – Global Innovation	ryan.spanier@kudelskisecurity.com
Amy Fleischer	Program Manager	amy.fleischer@kudelskisecurity.com
Ken Toler	Principal Application Security Manager	ken.toler@kudelskisecurity.com