2024. 10.30.

DE-FI LENDING PROTOCOL THREAT MODELING REPORTS

TEAM | DeepHigh

0. Abstract

Decentralized Finance (DeFi) lending protocols have revolutionized financial services but pose unique security challenges not adequately addressed by traditional threat modeling frameworks like STRIDE. In this paper, we introduce **De-FAULT**, a specialized threat modeling framework tailored for DeFi lending protocols. De-FAULT encompasses six critical categories: **Decentralized Issue**, **Coding Flaw**, **Access Control**, **Upgradeable Contract**, **Business Logic**, and **Tampered Ratio**, providing a structured approach to identify and mitigate vulnerabilities specific to DeFi lending. We conducted prior surveys, including an analysis of the Compound protocol and audit reports from various platforms, to inform our framework. Through comprehensive threat modeling—including diagramming, function identification, threat enumeration, and the development of a risk library—we established essential invariants that DeFi lending protocols must uphold. We applied De-FAULT and these invariants to eleven different DeFi lending protocols, such as Venus and Aave V3, to validate their effectiveness. Our findings demonstrate that protocols adhering closely to these invariants exhibit stronger security and are less susceptible to known attack vectors. The De-FAULT framework offers a practical tool for enhancing the security of DeFi lending platforms, addressing the unique risks of decentralized finance, and fostering greater trust and sustainable growth in the DeFi ecosystem.

1. Introduction

Decentralized Finance (DeFi) has revolutionized the financial industry by providing open and permissionless financial services on blockchain platforms. Among these services, DeFi lending protocols have gained significant popularity, allowing users to lend and borrow assets without traditional intermediaries. However, this rapid growth has been paralleled by a surge in security incidents, underscoring the critical need for effective threat modeling in the development of secure DeFi lending protocols.

Threat modeling is an essential process in software development and security engineering, enabling teams to proactively identify, evaluate, and mitigate potential security risks. In traditional computer security, frameworks like STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) are widely used to systematically analyze threats. These models have been effective in Web2 environments, where assets are primarily data and the attack surfaces are wellunderstood.

However, applying traditional threat modeling frameworks to Web3 and blockchain technologies poses significant challenges. The unique characteristics of blockchain, such as immutability, decentralized control, and the direct representation of monetary value, introduce new attack vectors and amplify the potential impact of vulnerabilities. Unlike Web2 applications, where assets are typically data, in Web3, assets often represent actual financial value, and security breaches can lead to immediate and irrevocable financial losses. Additionally, the techniques and resources available to attackers differ substantially, necessitating a tailored approach to threat modeling.

Despite the increasing frequency of security breaches in the DeFi space, there is a notable lack of standardized threat modeling frameworks specifically designed for Web3 and smart contracts. While existing resources, such as academic papers, online articles, and blogs provide valuable insights into smart contract vulnerabilities, they do not offer comprehensive frameworks that developers can utilize during the development phase to ensure code security.

DeFi lending protocols are among the most widely used services in the Web3 ecosystem. However, there is a scarcity of threat modeling frameworks specialized for these platforms. To address this gap, we introduce **De-FAULT**, a novel threat modeling framework tailored specifically for DeFi lending protocols. The De-FAULT framework is designed to assist developers and security professionals in systematically identifying potential threats and implementing effective mitigation strategies throughout the development lifecycle.

Our framework encompasses all stages of threat modeling—from diagramming system architectures to threat enumeration and mitigation planning. It focuses on key risk categories pertinent to DeFi lending protocols, including Decentralization Issues, Coding Flaws, Access Control weaknesses, Upgradeable Contract vulnerabilities, Business Logic errors, and Tampered Ratios.

Furthermore, we have identified fundamental invariants that any DeFi lending protocol must uphold to maintain security and integrity. To validate these invariants and demonstrate the practical applicability of the De-FAULT framework, we conducted an extensive analysis of 11 DeFi lending protocols, including Compound, Venus (Core and Isolated), Aave, Euler V2, and others. We assessed whether these protocols adhered to the identified invariants, examined differences in their implementations, and analyzed instances where lack of adherence led to vulnerabilities.

To further verify our threat modeling approach, we developed test cases based on the defined invariants and applied them to selected protocols such as Compound, Venus, Aave, and Euler. This empirical evaluation allowed us to identify potential weaknesses and validate the effectiveness of our framework in real-world scenarios.

In conclusion, the De-FAULT framework provides a comprehensive and specialized approach to threat modeling for DeFi lending protocols. By addressing the unique challenges of Web3 security, it serves as a vital tool for developers and security practitioners aiming to build secure and resilient DeFi platforms. Through systematic threat identification and mitigation, the framework contributes to enhancing the overall security posture of the DeFi ecosystem.

2. Prior Surveys

Before starting this research, we conducted some preliminary studies to build a solid understanding of DeFi lending protocols and to help develop our threat modeling framework.

2.1. Compound

Compound is a decentralized lending protocol that represents the basic functions of DeFi lending platforms. It allows users to supply assets to liquidity pools and earn interest, while borrowers can obtain assets by providing collateral. Due to its straightforward design, widespread use, and strong security record, Compound serves as a textbook example in the DeFi lending space.

As one of the most influential protocols, Compound has the second-highest Total Value Locked (TVL) among lending platforms, reflecting its wide use and trust within the community. Its codebase has been copied numerous times, serving as the foundation for many other lending protocols. Importantly, Compound has not experienced any major hacking incidents, which demonstrates its security and reliability.

To gain a comprehensive understanding of Compound, we examined its official documentation and analyzed its GitHub repository. The protocol works by allowing users to supply assets to its liquidity pools, receiving cTokens (e.g., cETH, cDAI) in return. These cTokens represent the user's share in the pool and automatically earn interest over time.

Borrowers can access these assets by providing collateral that exceeds the value of the borrowed assets, ensuring over-collateralization and reducing the risk of default.

Interest rates in Compound are dynamically adjusted based on the utilization rate of each asset pool. Higher utilization leads to higher interest rates, encouraging suppliers to provide more liquidity and balancing supply and demand within the market. The protocol enforces collateral factors for each asset, which determine the maximum amount that can be borrowed against the supplied collateral. If a borrower's collateral falls below the required maintenance threshold due to market changes or accrued interest, their position becomes eligible for liquidation. This mechanism ensures the protocol remains solvent by allowing others to repay a portion of the debt in exchange for a discount on the collateral.

Understanding the user flow and the interactions between smart contracts within Compound was crucial for our study. Users interact with various functions such as mint to supply assets, borrow to obtain loans, repayBorrow to repay debts, and redeem to withdraw supplied assets. The protocol's architecture involves several core smart contracts, including the Comptroller for risk management, cToken contracts for each supported asset, interest rate models, and the price oracle for real-time asset valuation. Our analysis of these user flows and functionalities was instrumental in creating diagrams during the threat modeling process and served as a foundation for our subsequent threat modeling efforts.

By thoroughly analyzing Compound's design, features, and operations, we established a foundational framework for evaluating other DeFi lending protocols. This deep understanding helped us identify potential security risks and informed the development of our threat modeling framework. Given its exemplary status and strong security track record, Compound served as an ideal benchmark for our research.

2.2. Analysis of Lending Protocol Audits via Solodit

In addition to studying Compound, we conducted a detailed analysis of lending protocol audit reports available on Solodit, a platform that gathers smart contract audits and security analyses. This analysis aimed to identify common vulnerabilities and security issues in DeFi lending protocols, providing real-world data to inform our threat modeling framework.

We collected and reviewed audit reports of various lending protocols from Solodit, ensuring a diverse representation of platforms with different designs and features.

Through careful examination, we identified vulnerabilities and security flaws reported by professional auditors. These vulnerabilities were systematically categorized based on their nature and the associated risks, such as coding errors, access control weaknesses, business logic flaws, and issues related to decentralization or upgradability.

The insights gained from this analysis were crucial in shaping our threat modeling framework, De-FAULT. By categorizing the vulnerabilities and understanding the common threats faced by DeFi lending protocols, we were able to tailor our framework to address the most critical security concerns. This categorization also helped in developing our risk library, which serves as a reference for potential threats and their mitigations.

This comprehensive survey revealed patterns in the types of vulnerabilities that frequently occur in DeFi lending protocols. For example, coding flaws like improper input validation or reentrancy vulnerabilities were common, as were issues related to insufficient access control and flawed business logic. Understanding these patterns allowed us to focus our threat modeling efforts on areas that are most susceptible to attacks.

By analyzing the audit reports, we also gained insights into the effectiveness of existing security measures and the importance of following best practices in smart contract development. This knowledge reinforced the need for a specialized threat modeling framework for DeFi lending protocols, as traditional security models do not fully address the unique challenges posed by blockchain technology and smart contracts.

In summary, the detailed analysis of lending protocol audits via Solodit provided valuable data that directly contributed to the development of our De-FAULT framework. It highlighted the common security issues within the DeFi lending space and underscored the necessity for a systematic approach to threat modeling tailored to the specific needs of Web3 applications.

3. De-FAULT: De-Fi Lending Protocol Threat Modeling Framework

Considering the distinct challenges specific to DeFi lending protocols and the inadequacy of existing threat modeling frameworks to address them, we propose **De-FAULT**, a specialized threat modeling framework tailored for DeFi lending protocols. Traditional frameworks like STRIDE do not fully capture the nuances of Web3 environments, where assets have intrinsic monetary value, and the attack surfaces differ significantly from traditional Web2 applications. Our goal with De-FAULT is to provide a systematic approach that helps developers and security professionals identify, categorize, and mitigate the specific threats inherent in DeFi lending protocols.

The De-FAULT framework focuses on six critical categories of potential threat elements: **Decentralized Issue**, **Coding Flaw**, **Access Control**, **Upgradeable Contract**, **Business Logic**, and **Tampered Ratio**. Each category addresses a distinct aspect of the protocol's design and implementation that could be exploited by malicious actors or lead to unintended consequences. This model covers key risk factors in a comprehensive yet efficient manner, ensuring complete protection without overwhelming complexity.

By adopting the De-FAULT framework, we aim to enhance the security and reliability of DeFi lending protocols. This framework not only assists in the proactive identification of potential threats during the development phase but also promotes the implementation of best practices in smart contract design. Ultimately, De-FAULT seeks to contribute to the overall robustness of the DeFi ecosystem, fostering greater trust among users and encouraging the sustainable growth of decentralized financial services.

- **Decentralized Issue:** Problems arising from insufficient decentralization in the protocol's governance or control mechanisms. This includes situations where sensitive functions are not managed by decentralized autonomous organizations (DAOs), where authority is overly centralized in a single entity, or where essential security practices like time locks and multi-signature wallets are not implemented. Such centralization can lead to abuse of power or create single points of failure within the system.
- **Coding Flaw:** Errors or vulnerabilities in the codebase, such as bugs, programming mistakes, or inadequate input validation. This encompasses issues like missing return values, incorrect use of inequality operators, improper documentation or comments, decimal precision errors, state variable synchronization mismatches or not using the latest values, gas-related issues, and vulnerabilities like reentrancy attacks. These flaws can be exploited to compromise the security, functionality, or performance of the protocol.
- **Access Control:** Flaws in the protocol's access control mechanisms, including inadequate authentication or authorization checks. Such vulnerabilities can allow unauthorized users to gain access to restricted functions or sensitive data, leading to potential misuse or exploitation.
- **Upgradeable Contract:** Risks associated with the upgradeable nature of smart

contracts, particularly when best practices are not followed in proxy patterns. This includes failures such as missing initialize functions, improper handling of storage gaps, or other mistakes that can introduce vulnerabilities during contract upgrades. These issues can lead to security breaches or unauthorized modifications that affect the integrity of the protocol.

- **Business Logic:** Vulnerabilities arising from flaws in the protocol's business logic. This involves incorrect implementation of operational rules or processes, such as missing necessary validations, or calculations that lack essential components. Such flaws can lead to unexpected behaviors, financial losses, or opportunities for exploitation by malicious actors.
- **Tampered Ratio:** Risks involving the manipulation or tampering of critical ratios or parameters used by the protocol, such as collateralization ratios, interest rates, or price feeds from oracles. Exploiting these can give attackers undue advantages, potentially leading to financial instability within the protocol.

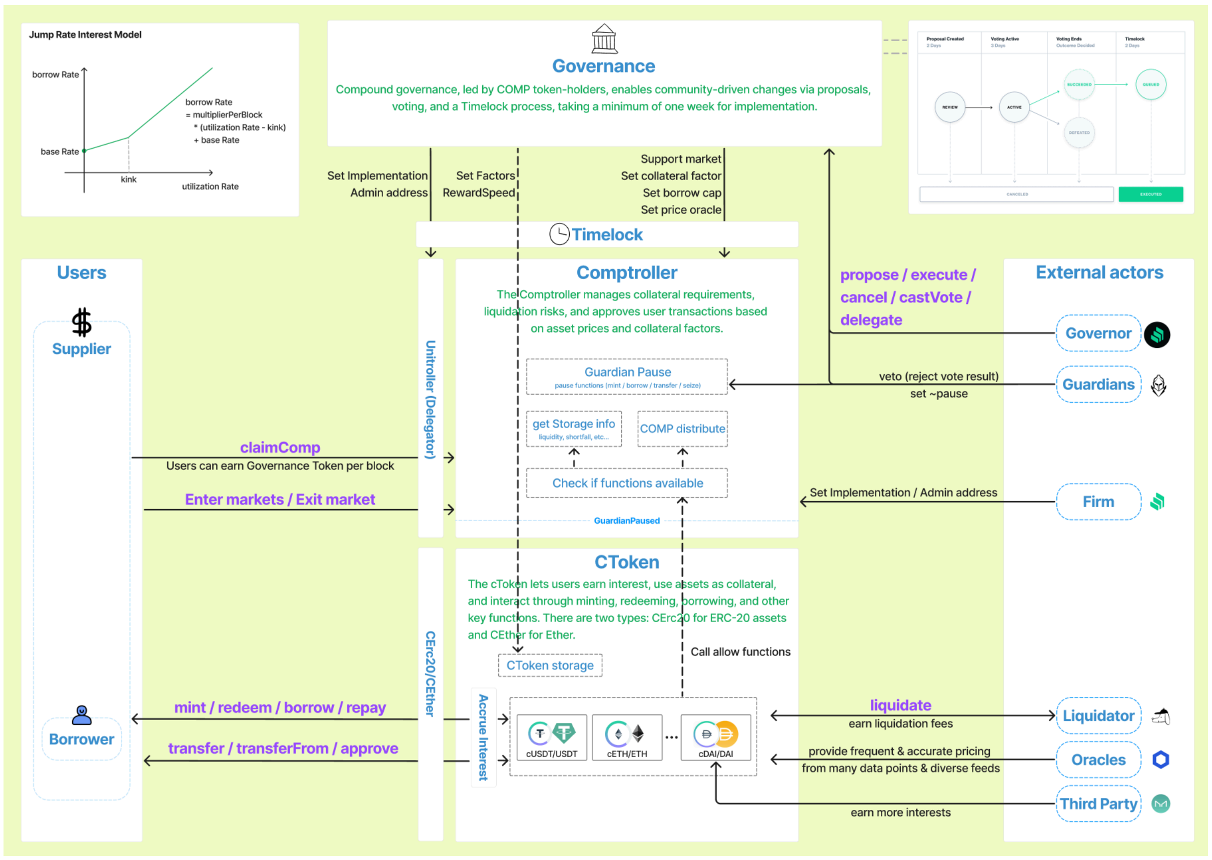
	Decentralized Issue	**Coding Flaw**	**Access Control**	**Upgradeable Contract**	**Business Logic**	**Tampered Ratio**
Deposit		O			O	O
Withdraw		O			O	O
Borrow		O			O	O
Repay		O			O	
Liquidation		O			O	O
Interest / Reward		O			O	
Oracle		O			O	O
Management	O	O	O	O	O	O
User State		O	O		O	
Protocol State		O	O		O	

4. Threat Modeling

Following the standard threat modeling steps, we conducted a comprehensive analysis to identify potential security threats within DeFi lending protocols. This process allowed us to systematically examine the protocols from multiple angles to uncover vulnerabilities and assess risks.

4.1. Diagramming

Using our prior analysis of Compound as a foundation, we created detailed diagrams to represent the architecture and user flows of a typical DeFi lending protocol. These diagrams included components such as user interfaces, smart contracts, external systems like price oracles, and governance mechanisms. By mapping out the interactions between these components, we were able to visualize how data and control flow through the system, which is crucial for identifying potential attack vectors and understanding the overall security posture of the protocol.


Jump Rate Interest Model
borrow Rate
borrow Rate = multiplierPerBlock * (utilization Rate - kink) + base Rate
base Rate
kink
utilization Rate
Governance
Compound governance, led by COMP token-holders, enables community-driven changes via proposals, voting, and a Timelock process, taking a minimum of one week for implementation.
Set Implementation
Admin address
Set Factors
RewardSpeed
Support market
Set collateral factor
Set borrow cap
Set price oracle
Timelock
Users
Supplier
Borrower
Comptroller
The Comptroller manages collateral requirements, liquidation risks, and approves user transactions based on asset prices and collateral factors.
Unitroller (Delegator)
Guardian Pause
pause functions (mint / borrow / transfer / seize)
get Storage info
liquidity, shortfall, etc...
COMP distribute
Check if functions available
GuardianPaused
claimComp
Users can earn Governance Token per block
Enter markets / Exit market
CToken
The cToken lets users earn interest, use assets as collateral, and interact through minting, redeeming, borrowing, and other key functions. There are two types: CErc20 for ERC-20 assets and CEther for Ether.
CErc20/CEther
CToken storage
Call allow functions
Accrue Interest
cUSDT/USDT
cETH/ETH
cDAI/DAI
mint / redeem / borrow / repay
transfer / transferFrom / approve
External actors
propose / execute / cancel / castVote / delegate
Governor
veto (reject vote result)
set ~pause
Guardians
Set Implementation / Admin address
Firm
liquidate
earn liquidation fees
Liquidator
provide frequent & accurate pricing from many data points & diverse feeds
Oracles
earn more interests
Third Party

4.2. Identification of Function

We identified the main functions of the DeFi lending protocol based on the diagramming phase. The key functions are as follows:

- **Deposit:** The deposit function allows users to supply assets to the lending protocol, earning interest or rewards in return. This function is foundational to liquidity provision and forms the basis for other lending and borrowing operations within the protocol.
- **Withdraw:** The withdraw function enables users to retrieve their deposited assets, including any accumulated interest. This function ensures liquidity remains accessible to depositors while maintaining protocol solvency.
- **Borrow:** The borrow function allows users to take loans by using their deposited assets as collateral. Borrowing introduces risk to the protocol, as collateralization ratios must be managed to prevent under-collateralized positions.
- **Repay:** The repay function lets users return borrowed funds to the protocol, closing out their debt. Repayment impacts protocol liquidity and risk exposure, as it affects the amount of assets available for other borrowers.
- **Liquidation:** The liquidation function enables the protocol to secure assets when borrowers' collateral falls below required thresholds. This function is critical in mitigating risks associated with under-collateralized loans and maintaining protocol stability.
- **Interest Model:** The interest model determines the rates at which users earn on deposits and pay on borrowings. It often adjusts dynamically based on supply and demand, impacting user incentives and overall protocol liquidity.
- **Oracle Data:** Oracle data provides external pricing information for assets within the protocol. Accurate and timely oracle data is essential for maintaining fair collateral valuations and preventing price manipulation vulnerabilities.
- **Protocol Management:** Protocol management encompasses the administrative functions that control parameters, deploy upgrades, and oversee protocol governance. This function is critical for maintaining security, adaptability, and compliance with community or governance standards.
- **User State:** User state tracks individual user actions, balances, collateral, and debt positions. It ensures accurate record-keeping for user accounts and underpins the security of each user's assets and liabilities within the protocol.
- **Protocol State:** Protocol state maintains global variables such as total liquidity,

outstanding debt, and interest rates. It serves as a snapshot of the protocol's overall health and is central to risk assessment and management, impacting the protocol’s ability to respond to market changes.

By identifying these functions, we established a clear understanding of the protocol's operational landscape. This step was essential for pinpointing where potential threats could emerge and for guiding the subsequent phases of our threat modeling process.

4.3. Risk Library

We compiled a comprehensive risk library consisting of 352 items, derived from our analysis of prior works. This included the examination of Solodit's DeFi lending audits and audit reports from various lending protocols such as Aave, Venus, Euler, and others. This risk library serves as the foundation for the subsequent stages of our threat modeling process, providing empirical evidence and real-world examples of vulnerabilities.

Each identified threat was categorized according to the De-FAULT framework, and we further grouped them into the following risk categories. Below, we expand on each category to provide a deeper understanding of the potential risks.

- **Theft of User Funds:** An attacker directly steals user funds or liquidity provided by users. This includes exploits that allow unauthorized withdrawals, draining of liquidity pools, or manipulation of balances to the attacker's benefit.
- **Theft of Unclaimed Yield:** An attacker steals unclaimed rewards like interest or yields, or prevents users from receiving them. This could involve intercepting reward distributions, altering the reward calculation mechanisms, or manipulating the timing of reward claims to their advantage.
- **Oracle Attack:** Manipulating oracle data to compromise the system. Attacks could involve feeding false price data to manipulate collateral valuations, leading to wrongful liquidations, under-collateralized loans, or arbitrage opportunities that unfairly benefit the attacker. This can be achieved through techniques like flash loan attacks to temporarily distort prices.
- **Account Takeover:** An attacker seizes administrative privileges to take over accounts. This includes exploiting weak access controls, compromised private keys, or vulnerabilities in authentication mechanisms to gain control over privileged accounts or governance functions, potentially allowing them to alter

protocol parameters or drain funds.

- **UX Violation:** User experience is compromised due to code and logic errors, without the presence of an attacker. Issues could include incorrect balance displays, transaction errors, or confusing interfaces leading to user mistakes. While not directly exploited by attackers, these issues can erode user trust and lead to unintended financial losses.
- **Market Risk:** Risks like fund freezing or increased debts occur due to code and logic errors, without the presence of an attacker. Examples include smart contract bugs causing assets to be locked indefinitely, interest rates being miscalculated leading to unsustainable debt accrual, or failure to handle extreme market conditions properly.
- **Grief Attack:** An attacker harms users or the system without direct personal gain. Such attacks may involve spamming the network to cause denial of service, triggering unnecessary protocol actions to burden the system, or manipulating certain functions to cause loss or inconvenience to other users, even if the attacker does not profit financially.
- **Dev/Maintenance Difficulties:** Difficulties in maintenance or risks during the development phase exist. This includes issues like poor code documentation, lack of modularity, complex code that is hard to audit or update, and inadequate testing procedures. Such problems can lead to overlooked vulnerabilities and hinder the protocol's ability to respond to emerging threats.

This categorization helps in systematically addressing each type of risk and forming appropriate mitigation strategies. By understanding the nature and impact of these risks, developers and security professionals can prioritize efforts to strengthen the protocol's security posture.

4.4. Threat Enumeration

Based on the identification of functions, we systematically identified potential threats at each point in the protocol, categorizing them using the De-FAULT framework. This process involved analyzing how each function could be exploited or fail due to various vulnerabilities.

By enumerating these threats, we can identify which threats or vulnerabilities require the most attention within each function. This structured approach helps us focus on the specific areas that need careful monitoring and ensures that we address the most critical vulnerabilities with targeted mitigation strategies in the next phase.

4.5. Risk Tree

We organized the identified threats into a risk tree, visually representing the relationships between threats, vulnerabilities, and their potential impact on the protocol. This hierarchical structure helped us understand how different threats are interconnected and how they could cascade to cause larger systemic issues. The risk tree served as a valuable tool for identifying root causes and potential single points of failure within the protocol.

ID	Risk Tree			
	1	Collateral Deposit		
	OR	1.1	Theft of User Funds	
RT1		OR	1.1.1	First depositor can manipulate share distribution
	OR	1.2	UX Violation	
RT2		OR	1.2.1	Users' fund loss due to excess collateral not refunded
RT3		OR	1.2.2	Permanent loss from minting tokens to the zero address
RT4		OR	1.2.3	Users' fund loss from lack of slippage protection during deposits
RT5		OR	1.2.4	Unexpected liquidations due to inability to adjust position sizes
RT6		OR	1.2.5	Unintended account status changes leading to unauthorized actions
	OR	1.3	Market Risk	
RT7		OR	1.3.1	Freezing of deposited funds impacting all user shares
RT8		OR	1.3.2	Inadequate validation of collateral leading to systemic risk
	2	Collateral Withdrawal		
	OR	2.1	Theft of User Funds	
RT9		OR	2.1.1	Unsafe external calls during withdrawal lead to theft of user funds
RT10		OR	2.1.2	Insufficient parameter checks allow attackers to withdraw others' collateral
RT11		OR	2.1.3	Attackers prevent position closure by manipulating liquidity
RT12		OR	2.1.4	Unauthorized fund transfers due to improper access control
	OR	2.2	UX Violation	
RT13		OR	2.2.1	Withdrawal of native assets with maximum amount causes transaction failure
RT14		OR	2.2.2	Logic and Calculation Errors Leading to Users Not Receiving Expected Assets
RT15		OR	2.2.3	Hard-coded slippage values prevent withdrawals during volatility
RT16		OR	2.2.4	Users cannot withdraw blacklisted or deprecated tokens

RT17		OR	2.2.5	Funds get stuck due to improper state updates during withdrawals
	3	Borrow		
	OR	3.1	Theft of User Funds	
RT18		OR	3.1.1	Flash Loan Abuse with Temporary Asset Ownership
RT19		OR	3.1.2	Reentrancy Attacks during Borrow Operations
	OR	3.2	Theft of Unclaimed Yield	
RT20		OR	3.2.1	Zero-Interest Borrowing Exploitation during Protocol Shutdown
	OR	3.3	Grief Attacks	
RT21		OR	3.3.1	Griefing Attacks via Manipulation of Minimal Amounts
	OR	3.4	Market Risk	
RT22		OR	3.4.1	Bypassing Individual Collateral Borrow Limits
RT23		OR	3.4.2	Incorrect Borrow Fee Calculations Affecting Protocol Revenue
RT24		OR	3.4.3	Borrowing Allowed for Users in Liquidation Positions
	4	Repay		
	OR	4.1	Theft of User Funds	
RT25		OR	4.1.1	State Mismatch Prevents Accurate Repayment
RT26		OR	4.1.2	Front-Running Leading to Unintended Repayment for Others
	OR	4.2	UX Violation	
RT27		OR	4.2.1	Misdirected Refunds Due to Incorrect Repayment Logic
	OR	4.3	Market Risk	
RT28		OR	4.3.1	Inability to Handle Bad Debt Post-Liquidation
RT29		OR	4.3.2	Oversupply of Assets Due to Missing Burn Mechanism
	OR	4.4	Grief Attack	
RT30		OR	4.4.1	Front-Running Repayments to Cause Transaction Reverts
	5	Liquidate		
	OR	5.1	Theft of User Funds	
RT31		OR	5.1.1	Over Liquidation Allowing the Liquidator to Avoid Full Payment
RT32		OR	5.1.2	Lack of Validation for Liquidation Orders
RT33		OR	5.1.3	Excessive Liquidator Profits Through Repeated Partial Liquidations
	OR	5.2	Theft of Unclaimed Yield	
RT34		OR	5.2.1	Loss of Premium Due to Front-Running
	OR	5.3	UX Violation	
RT35		OR	5.3.1	Transaction Failures Due to State Changes Between Approval and Execution
RT36		OR	5.3.2	Failure to Liquidate Due to Blacklisted or Restricted Assets

	OR	5.4	Market Risk	
RT37		OR	5.4.1	Locked Premium Due to Incorrect PnL Handling
RT38		OR	5.4.2	Liquidation Does Not Improve Borrower's Position
	OR	5.5	Grief Attack	
RT39		OR	5.5.1	Liquidators Forced to Pay More Due to Borrower Manipulation
RT40		OR	5.5.2	Liquidators Suffer Losses Due to Borrower Front-Running
	6	Protocol Management		
	OR	6.1	Account Takeover	
RT41		OR	6.1.1	Unprotected Critical Functions Allowing Unauthorized Access
RT42		OR	6.1.2	Centralized Control Leading to Single Point of Failure
RT43		OR	6.1.3	Improper Role Management Allowing Unauthorized Actions
	OR	6.2	Theft of User Funds	
RT44		OR	6.2.1	Reentrancy Attacks Due to Improper Function Logic
RT45		OR	6.2.2	Unprotected Initialization Functions Leading to Contract Hijacking
	OR	6.3	UX Violation	
RT46		OR	6.3.1	Lack of Input Validation Leading to Operational Failures
	OR	6.4	Market Risk	
RT47		OR	6.4.1	Improper Parameter Configuration Enables Exploitation
RT48		OR	6.4.2	Desynchronization Due to Improper State Management
	OR	6.5	Dev/Maintenance Difficulties	
RT49		OR	6.5.1	Storage Collisions in Upgradeable Contracts
RT50		OR	6.5.2	Chain-Specific Issues Not Accounted For
	OR	6.6	Grief Attack	
RT51		OR	6.6.1	Abuse of Allowlist to Block User Withdrawals
	7	Interest and Rewards		
	OR	7.1	Theft of Unclaimed Yield	
RT52		OR	7.1.1	Attackers Stealing Unclaimed Rewards from the Protocol
RT53		OR	7.1.2	Unfair Reward Distribution Due to MEV or Just-in-Time Deposits
RT54		OR	7.1.3	Unauthorized Claiming of Rewards on Behalf of Others
	OR	7.2	UX Violation	
RT55		OR	7.2.1	Incorrect Calculations Leading to Financial Discrepancies and Unintended Consequences
RT56		OR	7.2.2	Interest Accruing During Paused Repayment Periods
	OR	7.3	Market Risk	
RT57		OR	7.3.1	Incorrect Reward Calculations Leading to Inflated Values

RT58		OR	7.3.2	Financial Discrepancies Due to Improper Handling of Paused Functions
	8	Oracle		
	OR	8.1	Theft of User Funds	
RT59		OR	8.1.1	Flash Loan Manipulation of Oracle Data
	OR	8.2	Theft of Market Funds	
RT60		OR	8.2.1	Reentrancy Attacks Through Oracles
	OR	8.3	Market Risk	
RT61		OR	8.3.1	Manipulation of Oracle Data Using Low Liquidity Pools
RT62		OR	8.3.2	Oracle Using Outdated Data or Incorrect Parameters
RT63		OR	8.3.3	Lack of Validation on Oracle Data

4.6. Mitigation

For each identified threat, we proposed specific mitigation strategies aimed at addressing the root causes and enhancing the overall security of DeFi lending protocols. These strategies include:

ID	Risk Tree	Mitigations
RT1	First depositor can manipulate share distribution	- Handle the first liquidity provider's share separately to ensure fair distribution. - Increase the precision of numerical calculations to prevent disproportionate shares. - Have the development team act as the first depositor to set initial ratios correctly. - Burn a portion of the initial LP tokens to prevent manipulation (e.g., Uniswap V2 method). - Validate LP token amounts for subsequent depositors to ensure fair allocation.
RT2	Users' fund loss due to excess collateral not refunded	- Implement contract logic to automatically refund any excess collateral to the user during the deposit process. - Ensure accurate calculation of the required collateral amount and return any surplus funds immediately. - Include protective measures against front-running by securing the transaction order or using anti-front-running techniques.
RT3	Permanent loss from minting tokens to the zero address	- Add validation checks in all minting functions to prevent tokens from being minted to the zero address. - Ensure that all token transfers and mints verify that the recipient address is valid and not the zero address.

		- Implement thorough input validation and error handling to alert users of invalid addresses before transaction execution.
RT4	Users' fund loss from lack of slippage protection during deposits	- Allow users to set acceptable slippage limits on deposit and trade functions to cap potential losses.
RT5	Unexpected liquidations due to inability to adjust position sizes	- Provide functionality for users to adjust their positions without closing and reopening them. - Implement features that help users manage exposure and avoid liquidation. - Offer tools for monitoring positions and setting alerts for critical thresholds.
RT6	Unintended account status changes leading to unauthorized actions	- Implement strict access control checks in all functions modifying account statuses. - Ensure state changes occur only under correct conditions with proper authorization. - Regularly audit and test access control mechanisms.
RT7	Freezing of deposited funds impacting all user shares	- Isolate the impact of freezing new deposits from existing funds. - Implement logic to allow withdrawals even when deposits are paused. - Communicate system statuses transparently to users.
RT8	Inadequate validation of collateral leading to systemic risk	- Maintain a whitelist of approved collateral assets. - Enforce checks to ensure only authorized assets can be deposited. - Regularly review and update the list of acceptable collateral.
RT9	Unsafe external calls during withdrawal lead to theft of user funds	- Implement safe external call practices by checking the success of external calls and handling failures appropriately. - Validate all input parameters before making external calls to ensure they are as expected. - Verify and handle return values of external calls to manage unexpected results. - Interact only with whitelisted or trusted contracts to minimize risks associated with untrusted external calls. - Use access control mechanisms to restrict external calls to authorized addresses. - Employ reentrancy guards and follow secure coding patterns like the Checks-Effects-Interactions pattern to prevent reentrant attacks.
RT10	Insufficient parameter checks allow attackers to withdraw others' collateral	- Enforce strict access controls to ensure that withdrawal functions can only be called by the owner of the collateral or authorized parties.

		- Add validation checks to confirm that all input parameters correspond to the caller's own assets. - Verify that token IDs, position IDs, or account addresses match the records associated with the caller. - Maintain accurate mappings of user ownership for collateral and positions, and reference these during withdrawal processes. - Regularly audit the contract code to identify and rectify any missing validation checks. - Implement comprehensive unit and integration tests to cover scenarios where an attacker might attempt to withdraw funds belonging to others.
RT11	Attackers prevent position closure by manipulating liquidity	- Remove the total current liquidity associated with the position during closure, not just the initial amount. - Reference up-to-date state variables to account for any changes in liquidity. - Implement safeguards to prevent unauthorized users from modifying others' positions.
RT12	Unauthorized fund transfers due to improper access control	- Implement robust access control to prevent unauthorized fund transfers. - Ensure that withdrawals to sanctioned addresses are redirected to secure holding accounts or handled according to compliance requirements. - Continuously monitor user statuses to prevent processing withdrawals from newly sanctioned accounts. - Update withdrawal logic to handle edge cases involving sanctioned or restricted addresses.
RT13	Withdrawal of native assets with maximum amount causes transaction failure	- Implement checks to correctly handle native assets and special values. - Use hardcoded decimal values where appropriate for native assets. - Ensure functions are compatible with native assets and do not rely on token interface methods that are not applicable. - Test edge cases involving maximum values and native assets.
RT14	Logic and Calculation Errors Leading to Users Not Receiving Expected Assets	- Synchronize user inputs with the contract's current state values. - Recalculate final amounts after adjustments and rounding to ensure accuracy. - Place critical success checks before function returns to ensure operations completed. - Implement thorough testing and validation of calculations

		and logic. - Provide clear error messages and transparency in operations.
RT15	Hard-coded slippage values prevent withdrawals during volatility	- Allow users to specify their own acceptable slippage or price tolerance levels. - Implement dynamic slippage mechanisms that adjust based on market conditions. - Avoid hard-coding slippage values; make them configurable and flexible. - Provide users with tools to manage slippage settings.
RT16	Users cannot withdraw blacklisted or deprecated tokens	- Design the protocol to allow users to withdraw their collateral even if the token is blacklisted or deprecated. - Implement mechanisms to handle such tokens, like providing alternative withdrawal options. - Communicate with users about token status changes and available actions. - Ensure that blacklisting a token does not prevent users from accessing their own assets.
RT17	Funds get stuck due to improper state updates during withdrawals	- Ensure accurate state management by correctly updating both in-memory and storage variables. - Implement proper synchronization mechanisms to prevent inconsistencies. - Thoroughly test withdrawal logic, including edge cases like expired requests. - Provide clear communication to users regarding the status of their withdrawal requests.
RT18	Flash Loan Abuse with Temporary Asset Ownership	- Implement verification mechanisms that ensure long-term ownership of assets rather than just temporary possession. - Introduce time-based checks or require multiple block confirmations to validate ownership. - Restrict certain sensitive functions from being executed within the same transaction as the asset acquisition. - Analyze protocol logic to identify and mitigate potential flash loan exploit vectors.
RT19	Reentrancy Attacks during Borrow Operations	- Apply reentrancy guards to all external functions that modify state or transfer funds. - Follow the Checks-Effects-Interactions pattern to prevent reentrancy vulnerabilities. - Conduct thorough code audits to identify and fix potential reentrancy issues. - Use safe external calls and ensure that state changes occur before external calls.

RT20	Zero-Interest Borrowing Exploitation during Protocol Shutdown	- Ensure that borrowing functions are completely disabled when the market is closed. - Implement checks in borrow functions to verify that the protocol is active. - Update the protocol state appropriately during shutdown to prevent any borrowing. - Regularly test shutdown procedures to ensure all functionalities are appropriately restricted.
RT21	Griefing Attacks via Manipulation of Minimal Amounts	- Adjust financial calculations to properly account for dust amounts, ensuring they do not adversely affect outcomes. - Set minimum transaction thresholds to prevent exploitation using negligible amounts. - Exclude insignificant amounts from profit and loss calculations where appropriate. - Monitor for unusual activity patterns that may indicate griefing attempts and respond accordingly.
RT22	Bypassing Individual Collateral Borrow Limits	- Enforce borrow limit checks during all relevant operations, including both borrowing and collateral withdrawal. - Ensure that after any user action, their account remains within the safe borrowing limits defined by the protocol. - Implement comprehensive validation and risk assessment before processing transactions that affect borrowing power. - Regularly audit and test the enforcement mechanisms for borrow limits to ensure they function correctly.
RT23	Incorrect Borrow Fee Calculations Affecting Protocol Revenue	- Correct and verify the fee calculation formulas to ensure accurate fee collection. - Properly initialize all variables and constants used in fee computations before deployment. - Conduct regular audits of financial functions to verify accuracy and alignment with the protocol's economic models. - Maintain transparency with users regarding fee structures and provide detailed breakdowns of fee calculations.
RT24	Borrowing Allowed for Users in Liquidation Positions	- Include collateral ratios in the calculation of collateral value to accurately assess borrowing capacity. - Prevent users in or near liquidation from borrowing more funds. - Regularly update and verify collateral valuations to reflect accurate market conditions. - Implement safeguards to stop further borrowing when users reach critical risk thresholds.
RT25	State Mismatch Prevents Accurate	- Ensure state synchronization by updating all relevant

	Repayment	balances and status values immediately after repayment operations. - Implement validation checks to confirm that balances are accurately reflected post-repayment. - Conduct thorough audits of state management in the repayment logic to detect potential synchronization issues.
RT26	Front-Running Leading to Unintended Repayment for Others	- Limit repayment amounts to the actual borrowed amount by enforcing a cap on repayment transactions. - Implement front-running protection mechanisms, such as transaction ordering and delays, to prevent malicious timing attacks. - Add logic to ensure that a user cannot repay more than the exact debt they owe.
RT27	Misdirected Refunds Due to Incorrect Repayment Logic	- Ensure that refund mechanisms return excess funds to the correct user, typically the sender. - Implement accurate tracking of user addresses and proper handling within refund logic. - Thoroughly test all repayment and refund scenarios to prevent misdirection of funds. - Provide transparent transaction records for users to verify fund movements.
RT28	Inability to Handle Bad Debt Post-Liquidation	- Implement mechanisms to manage and resolve bad debt post-liquidation, such as secondary recovery processes or allowing partial debt repayment options. - Ensure that the protocol can handle scenarios where collateral is insufficient to fully cover the debt. - Regularly audit the liquidation process to identify and resolve edge cases involving unresolved debt.
RT29	Oversupply of Assets Due to Missing Burn Mechanism	- Implement a burn mechanism that removes repaid assets from circulation to prevent oversupply. - Ensure that the repayment process includes steps to reduce the circulating supply in proportion to the amount repaid. - Conduct regular checks on the asset supply to ensure stability and prevent inflation or devaluation.
RT30	Front-Running Repayments to Cause Transaction Reverts	- Modify repayment functions to handle overpayments gracefully by capping the repayment at the outstanding debt without causing a revert. - Implement checks that adjust the repayment amount if the debt has changed since the transaction was initiated. - Include anti-front-running measures, such as time delays or requiring user confirmations, to protect users from such attacks.

RT31	Over Liquidation Allowing the Liquidator to Avoid Full Payment	- Strengthen the checks on USDC balances during the liquidation process to ensure that liquidators cannot bypass their payment obligations. - Implement reentrancy guards to prevent callback functions from interfering with the liquidation process. - Regularly audit the liquidation functions to ensure they handle over-liquidation scenarios properly and fairly.
RT32	Lack of Validation for Liquidation Orders	- Add validation logic to ensure that every asset in a liquidation order batch is listed in the appropriate market. - Implement input verification to check the market status of each asset before processing the liquidation. - Regularly audit the liquidation process to ensure that batch orders are handled correctly and securely.
RT33	Excessive Liquidator Profits Through Repeated Partial Liquidations	- Set maximum discount rates for liquidations to prevent exploitation. - Limit the frequency or number of partial liquidations on a single position. - Implement fair liquidation mechanisms that balance incentives for liquidators and protect borrowers from undue loss.
RT34	Loss of Premium Due to Front-Running	- Set a minimum discount rate for liquidation to prevent front-running attacks from negating the liquidator's premium. - Cap the amount a liquidator can pay to limit their exposure in case of front-running. - Introduce time-locks or other transaction ordering mechanisms to prevent sudden changes in collateral just before liquidation.
RT35	Transaction Failures Due to State Changes Between Approval and Execution	- Accept exact amounts rather than shares in liquidation functions to reduce reliance on changing state variables. - Implement real-time checks and updates to ensure accurate values are used at execution time. - Provide clear error messages and guidance to users when transactions fail due to state changes.
RT36	Failure to Liquidate Due to Blacklisted or Restricted Assets	- Allow partial liquidation to avoid complete transaction failures when encountering restricted assets. - Implement fallback mechanisms to handle blacklisted or restricted assets, such as converting them to other tokens or using alternative transfer methods. - Regularly assess the impact of restricted assets on liquidation processes and update the protocol accordingly.

RT37	Locked Premium Due to Incorrect PnL Handling	- Remove restrictive conditions on net PnL so that premiums are distributed regardless of the specific PnL value. - Implement checks to ensure that premiums are appropriately handled and distributed in all scenarios. - Review the liquidation logic to ensure it does not inadvertently lock funds under unfavorable PnL conditions.
RT38	Liquidation Does Not Improve Borrower's Position	- Implement logic to ensure that liquidation reduces the borrower's debt and improves their position. - Prevent scenarios where liquidation can worsen the borrower's financial state. - Monitor liquidation outcomes to ensure they align with intended market stability goals.
RT39	Liquidators Forced to Pay More Due to Borrower Manipulation	- Limit the number of markets a borrower can enter, especially when they have outstanding debts. - Optimize liquidation processes to handle multiple markets efficiently without excessive gas costs. - Implement safeguards against borrower actions that could unfairly burden liquidators.
RT40	Liquidators Suffer Losses Due to Borrower Front-Running	- Implement mechanisms to protect liquidators, such as setting minimum premiums or allowing them to specify acceptable conditions. - Use anti-front-running techniques to prevent borrowers from disrupting liquidation processes. - Allow liquidators to cancel or adjust their transactions if conditions change adversely.
RT41	Unprotected Critical Functions Allowing Unauthorized Access	- Implement strict access control mechanisms to ensure only authorized entities can execute critical functions. - Use role-based access control (RBAC) patterns to manage permissions. - Protect administrative accounts with multi-signature wallets or time-lock mechanisms. - Regularly audit access controls and critical functions for vulnerabilities.
RT42	Centralized Control Leading to Single Point of Failure	- Decentralize control by implementing DAO-based governance structures. - Use multi-signature wallets requiring multiple approvals for critical actions. - Introduce time-lock mechanisms on administrative functions to allow community oversight and reaction time. - Regularly rotate keys and monitor for suspicious activities.
RT43	Improper Role Management Allowing	- Implement secure role management with proper grant and

	Unauthorized Actions	revoke processes. - Require multi-party consensus for assigning critical roles. - Regularly audit role assignments and monitor for unauthorized changes. - Use standardized access control libraries and follow best practices.
RT44	Reentrancy Attacks Due to Improper Function Logic	- Follow the Checks-Effects-Interactions pattern to structure functions securely. - Use reentrancy guards (e.g., nonReentrant modifiers) on functions that interact with external contracts. - Carefully audit all external calls within administrative functions. - Minimize external calls in critical functions or ensure they are made to trusted contracts.
RT45	Unprotected Initialization Functions Leading to Contract Hijacking	- Protect initialization functions so they can only be called once and by authorized parties. - Use standardized patterns for contract initialization, such as OpenZeppelin's Initializable contract. - Disable initializers in constructors using _disableInitializers() to prevent unauthorized reinitialization. - Audit and test initialization logic thoroughly.
RT46	Lack of Input Validation Leading to Operational Failures	- Implement comprehensive input validation on all administrative and configuration functions. - Enforce upper and lower bounds on critical parameters like fees, interest rates, and collateral factors. - Include checks for zero addresses, duplicate entries, and invalid data formats. - Employ defensive programming practices and extensive unit testing.
RT47	Improper Parameter Configuration Enables Exploitation	- Enforce validation and bounds on parameter settings during updates. - Require governance approval or multi-signature consensus for changing critical parameters. - Use upper and lower limits to prevent extreme or harmful values. - Regularly review and adjust parameters in line with market conditions.
RT48	Desynchronization Due to Improper State Management	- Ensure that all state changes are propagated correctly throughout the protocol. - Avoid caching critical variables that may change; fetch the latest values when needed. - Implement mechanisms to update cached values

		appropriately if caching is necessary. - Thoroughly test configuration changes in various scenarios to detect potential desynchronization issues.
RT49	Storage Collisions in Upgradeable Contracts	- Reserve storage gaps (unused storage slots) in contracts to allow for future variable additions without affecting existing storage layout. - Use standardized upgradeable contract patterns provided by libraries like OpenZeppelin. - Carefully manage and document storage layouts during upgrades. - Conduct storage layout audits before deploying upgrades.
RT50	Chain-Specific Issues Not Accounted For	- Make chain-dependent variables immutable and set them appropriately during deployment. - Account for differences in block time or other chain parameters in the protocol design. - Use time-based measurements instead of block numbers when appropriate. - Test the protocol on each target chain to ensure compatibility.
RT51	Abuse of Allowlist to Block User Withdrawals	- Implement decentralized management structures to prevent abuse of administrative privileges. - Establish checks and balances on administrative powers, possibly through community governance. - Limit the scope of allowlist functionalities to essential use cases. - Provide transparent processes and appeal mechanisms for users.
RT52	Attackers Stealing Unclaimed Rewards from the Protocol	- Ensure that when positions are closed, all due rewards are properly refunded to the users. - Implement mechanisms to prevent attackers from claiming rewards belonging to others. - Regularly audit reward distribution functions to ensure proper handling of unclaimed rewards. - Validate that rewards are correctly accounted for and distributed upon position changes.
RT53	Unfair Reward Distribution Due to MEV or Just-in-Time Deposits	- Modify reward distribution logic to factor in the duration of stake or implement anti-whale measures. - Use snapshot mechanisms to record balances before reward distribution. - Use private transactions or flashbots to prevent front-running and MEV exploits. - Implement time-weighted reward calculations.

RT54	Unauthorized Claiming of Rewards on Behalf of Others	- Add access control checks to ensure users can only claim their own rewards. - Implement proper authentication and authorization mechanisms in reward claiming functions. - Educate users about the risks and proper usage of claiming functions.
RT55	Incorrect Calculations Leading to Financial Discrepancies and Unintended Consequences	- Use up-to-date data in all calculations by calling functions that retrieve the latest values. - Ensure that interest is accrued and state variables are updated before performing any assessments or payouts. - Implement thorough testing and code reviews to catch errors in calculations and distribution functions. - Add checks within functions to verify that transfers have been successfully executed. - Provide clear communication to users about potential risks and how calculations are performed. - Maintain transparency in reward calculations and provide users with access to transaction records.
RT56	Interest Accruing During Paused Repayment Periods	- Implement logic to halt interest accrual when repayment functions are paused. - Allow users to repay debts even when other functions are paused, ensuring they can manage their obligations. - Clearly communicate to users how pauses affect their loans and interest. - Design the protocol to avoid penalizing users during maintenance or emergency pauses.
RT57	Incorrect Reward Calculations Leading to Inflated Values	- Implement strict validation when adding new tokens to ensure only intended tokens are included in rewards. - Adjust calculation methods to exclude balances that should not contribute to rewards (e.g., tokens sent directly to the contract). - Set up safeguards against manipulation of reward mechanisms by malicious actors. - Monitor reward distributions and adjust algorithms to maintain market equilibrium.
RT58	Financial Discrepancies Due to Improper Handling of Paused Functions	- Adjust calculation logic to exclude periods when functions were paused. - Ensure that interest and rewards are accurately calculated based on active periods only. - Implement controls to prevent manipulation when pausing and restarting functions. - Clearly communicate to users how pauses and resumptions

		affect their obligations and rewards. - Design the protocol to avoid financial burdens on users due to paused functions.
RT59	Flash Loan Manipulation of Oracle Data	- Use time-weighted average prices (TWAP) to resist flash loan price manipulation. - Implement oracle checks that prevent short-term price spikes from affecting critical operations. - Utilize decentralized oracles that aggregate data from multiple sources to ensure data reliability.
RT60	Reentrancy Attacks Through Oracles	- Implement reentrancy guards in all external function calls. - Use official libraries or well-reviewed contracts that include reentrancy protections. - Conduct thorough audits of all oracle-related contracts, ensuring they handle external calls securely.
RT61	Manipulation of Oracle Data Using Low Liquidity Pools	- Implement safeguards to avoid sourcing prices from low-liquidity pools. - Use oracles that rely on well-established, high-liquidity pools to avoid price manipulation. - Use median or TWAP mechanisms to stabilize price feeds in the event of manipulation.
RT62	Oracle Using Outdated Data or Incorrect Parameters	- Implement checks for data staleness and ensure oracles are consistently updated. - Set custom parameters such as maxDelayTime to ensure only recent data is used. - Include fallback mechanisms that switch to alternative oracles if data becomes outdated.
RT63	Lack of Validation on Oracle Data	- Always use current and recommended oracle interfaces that provide detailed data, including timestamps and round completeness. - Implement thorough validation of oracle responses, ensuring data is fresh and valid before usage. - Set up alerts or safeguards to detect and respond to anomalies or inconsistencies in oracle data promptly.

By implementing these mitigations, we aim to reduce the overall risk and strengthen the security of DeFi lending protocols. Proactive threat modeling and the application of targeted security measures are essential for fostering trust and ensuring the sustainable growth of the DeFi ecosystem.

5. De-Fi Lending Protocol Invariants

Through our comprehensive threat modeling process, we have identified a set of fundamental invariants that DeFi lending protocols must uphold to ensure security, stability, and integrity across their operations. These invariants serve as essential conditions that should always remain true, acting as a foundation for the correct functioning of the protocol's various components. By proposing these invariants in the form of a checklist, we aim to provide developers and auditors with a practical tool to verify the robustness of lending protocols during development and assessment.

This framework of invariants is inspired by the validation logic of the Compound protocol, known for its reliability and security in the DeFi space. We applied and validated these invariants across other DeFi protocols to ensure consistent reliability and functionality. By setting these invariants as a baseline, we examined how each invariant behaves within different protocols and identified any significant deviations in implementation. This comparative analysis highlights areas where protocols may need to strengthen their adherence to these critical conditions.

Invariants are conditions that must always hold true to maintain the protocol's stability and integrity. Violations of these invariants can lead to vulnerabilities, financial losses, or systemic failures. The following is a detailed list of invariants categorized by the primary functions within DeFi lending protocols.

- **Deposit**
    - ✓ Collateral deposits must be disallowed when the protocol is in a paused state.
    - ✓ The collateral market must exist to enable deposits.
    - ✓ The block state of the collateral market must be updated to the latest status.
- **Withdrawal**
    - ✓ The market for withdrawing collateral must be available.
    - ✓ Withdrawals must be limited to maintain Loan-to-Value (LTV) ratios.
    - ✓ The block state of the collateral withdrawal market must reflect the latest status.
    - ✓ Withdrawal amounts must not exceed the market's total collateral balance.
    - ✓ Users can only be removed from the debt list if their borrowings are within

LTV limits relative to collateral.

- **Borrow**
  - ✓ Loans must not be allowed when the protocol is paused.
  - ✓ The borrowing market must exist.
  - ✓ The borrower must be registered in the respective market.
  - ✓ Post-loan, total borrows in the market should not surpass the borrow cap.
  - ✓ Borrowers cannot exceed their LTV ratio relative to collateral.
  - ✓ The target market's block state must reflect the latest status before borrowing.
  - ✓ The borrowed amount must be less than the total available balance in the market.
- **Repay**
  - ✓ The repayment market must exist.
  - ✓ The repayment market's block state must be updated to reflect the latest status.
  - ✓ Users cannot be removed from the debt list if they still have outstanding balances.
  - ✓ Repaying more than the borrowed amount is disallowed.
- **Liquidation**
  - ✓ The market for both the borrowed and collateral assets must exist.
  - ✓ Only borrowers whose LTV exceeds limits, indicating a liquidity shortfall, are eligible for liquidation.
  - ✓ Repayment by the liquidator must not exceed the borrower's close factor.
  - ✓ Both collateral and borrowed assets' markets must be updated to the latest block state.
  - ✓ The liquidator and borrower cannot be the same account.
  - ✓ The repayment amount by the liquidator cannot be zero or exceed data type limits.
  - ✓ Collateral received by the liquidator must be less than the borrower's total collateral balance.
  - ✓ Both collateral and borrowed assets must be governed by the same administrative entity.
- **Management**
  - ✓ Interest calculations for reserve adjustments must reflect the latest state.

- ✓ When reducing reserves, adequate cash must be available in the protocol.
- ✓ Reserve reduction amounts must not exceed the current total reserves.
- ✓ Only authorized administrators may modify critical settings and parameters (e.g., admin changes, interest models).
- ✓ Only authorized administrators may suspend or resume protocol operations.
- ✓ Protocol initialization is restricted to a single execution, with reinitialization prevention.
- ✓ Only the owner can call the initialize function.
- ✓ Non-reentrancy mechanisms (e.g., Checks-Effects-Interaction (CEI) pattern) must be enforced on critical function calls.
- ✓ Only authorized administrators can set a new implementation in upgradable protocols.
- ✓ Only authorized administrators may appoint new admins when transferring administrative rights.
- ✓ Reserved storage (__gap) for upgrades and expansions must remain protected and unmodified.
- ✓ Upgrades must be conducted through a DAO and governance framework.
- ✓ Ownership transfers must follow a two-step process.

- **Interest and Rewards**
  - ✓ Borrow interest rates must not exceed the specified maximum.
  - ✓ During interest calculations, related variables (total reserves, borrows, and market indices) must be updated.
  - ✓ Interest calculations must reflect the latest state.
- **Oracle**
  - ✓ The underlying asset price from the oracle must not be zero; transactions must be halted if the price is zero.
  - ✓ Ensure the data reflects the latest status.

By adhering to these invariants, DeFi lending protocols can significantly reduce the risk of vulnerabilities and enhance their overall security posture. These invariants serve as a practical checklist for developers, auditors, and security professionals to verify the integrity of protocol implementations. In the following sections, we will apply this framework to various DeFi lending protocols, examining their adherence to these invariants and identifying areas for improvement.

6. Applying the Framework

To validate the effectiveness of the De-FAULT threat modeling framework and the set of invariants we proposed, we applied them to several DeFi lending protocols. This practical application involved conducting audits, developing test cases based on our threat modeling, and assessing each protocol's adherence to the identified invariants. By doing so, we aimed to determine whether our framework could accurately identify potential vulnerabilities and whether deviations from the invariants could lead to security issues.

We selected a diverse range of protocols, including well-established platforms like Aave V3, Compound, and emerging ones like Euler V2 and Venus. For each protocol, we performed a thorough analysis by:

- **Conducting Audits**: We reviewed the protocol's codebase, documentation, and previous audit reports to understand its design and implementation details.
- **Developing Test Cases**: Based on our threat modeling and the defined invariants, we wrote specific test cases to simulate various scenarios, including edge cases and potential attack vectors.
- **Assessing Invariant Compliance**: We checked whether each protocol adhered to the proposed invariants, noting any deviations or unique implementations.
- **Analyzing Implementation Differences**: When invariants were implemented differently or not at all, we analyzed these differences to understand their impact on the protocol's security and functionality.
- **Identifying Potential Vulnerabilities**: We evaluated whether deviations from the invariants could lead to vulnerabilities, assessing the likelihood and potential impact of such security issues.

Our analysis revealed that while many protocols adhered to the fundamental invariants, there were notable differences in implementation strategies. Some protocols introduced additional security measures or alternative mechanisms to achieve similar security goals. In cases where invariants were not strictly followed, we identified potential vulnerabilities that could be exploited under certain conditions.

By applying the De-FAULT framework and testing for invariants compliance across multiple protocols, we demonstrated the utility and effectiveness of our approach in real-world scenarios. The framework proved valuable in identifying potential weaknesses and guiding developers toward best practices in protocol design and implementation.

In the following subsections, we provide a brief overview of each protocol analyzed and discuss the specific findings related to the application of our framework.

6.1. Aave V3

Aave V3 is the latest version of the Aave protocol, a decentralized non-custodial liquidity market where users can participate as depositors or borrowers. It enhances the decentralized lending experience with new features aimed at improving efficiency, security, and cross-chain functionality. Aave V3 introduces cross-chain portals, allowing seamless movement of assets across different blockchain networks. It also features an Isolation Mode, enabling the protocol to list new assets while isolating their risk to protect the overall system. The Efficiency Mode (eMode) increases capital efficiency by allowing users to achieve higher borrowing power when supplying and borrowing correlated assets. Additionally, Aave V3 includes gas optimizations to reduce transaction costs and implements enhanced risk management through improved parameters and asset listings. These innovations make Aave V3 a significant evolution in decentralized lending.

6.2. Venus

Venus Protocol is a decentralized lending and borrowing platform on the Binance Smart Chain (BSC), integrating money market and stablecoin functionalities. It provides fast and low-cost decentralized financial services, allowing users to supply and borrow cryptocurrencies with algorithmically adjusted interest rates. Venus enables users to mint VAI, a synthetic stablecoin pegged to the U.S. dollar, using their collateral. Its Isolated Pool feature allows certain assets to be managed separately from the main pool, reducing risk exposure and supporting a wider range of assets. Venus leverages the speed and low fees of BSC, and uses the XVS token for community governance, enabling users to vote on protocol proposals and influence its development.

6.3. Euler V2

Euler Finance V2 is a non-custodial DeFi protocol that allows users to lend and borrow almost any crypto asset. It offers permissionless lending and borrowing markets with innovative risk management features. Euler V2 enables users to create lending markets for any ERC-20 token without requiring approval, fostering a more inclusive financial ecosystem. The protocol implements risk-adjusted collateral factors through isolation tiers, effectively managing collateral risk and protecting lenders. Euler V2 uses dynamic, reactive interest rates that adjust based on market utilization, promoting efficient capital allocation. It also features efficient liquidation mechanisms, such as Dutch auctions, to minimize losses during liquidations. With its sub-account feature, users can manage multiple positions under one account, enhancing flexibility and user experience.

6.4. BIFI

BiFi (Bifrost Finance) is a multi-chain DeFi project built on the BIFROST protocol, aiming to connect capital markets across different blockchains. It enables cross-chain lending and borrowing, bridging assets from multiple blockchains into a unified DeFi platform. BiFi leverages BIFROST's middleware technology for interoperability, allowing users to lend and borrow assets across various blockchains seamlessly. The protocol is designed to handle large volumes of transactions across chains, aiming to reduce costs and improve transaction speeds. BiFi focuses on providing a user-friendly interface, making it accessible for users interacting with multiple blockchain networks and simplifying the cross-chain DeFi experience.

6.5. Inverse Finance

Inverse Finance is a DeFi platform offering a suite of financial tools, including lending, borrowing, and stablecoin minting, governed by a decentralized autonomous organization (DAO). It provides decentralized lending and stablecoin services with a focus on community governance and fixed-rate loans. Users can mint DOLA, a USD-pegged stablecoin, by depositing collateral into the protocol. Inverse Finance offers markets for various cryptocurrencies and provides fixed-rate borrowing through its Anchor platform, allowing users to plan with predictable costs. The INV token is used for governance, enabling token holders to propose and vote on protocol changes, thus directly influencing the platform's development. The platform also implements yield aggregation strategies to maximize returns on deposited assets.

6.6. Beta Finance

Beta Finance is a permissionless money market protocol designed for lending, borrowing, and short-selling crypto assets. It empowers users to create money markets for any token and enables decentralized short-selling, providing tools for hedging and speculation. Beta Finance allows anyone to start lending and borrowing markets for new tokens without requiring approval, fostering financial inclusivity. It offers an isolated collateral model, reducing systemic risk by isolating collateral for each position. The protocol employs automated risk management algorithms to handle risks associated with volatile assets. Users can stake BETA tokens to participate in protocol governance, influencing decisions and future developments.

6.7. BendDAO

BendDAO is a decentralized peer-to-pool protocol facilitating NFT-backed loans and liquidity provision. It allows users to obtain instant liquidity using NFTs as collateral and earn interest by supplying ETH to the protocol. Borrowers can use supported NFTs to borrow ETH, providing a way to access funds without selling their NFTs. The protocol has an ETH liquidity pool where depositors earn interest, incentivizing liquidity provision. BendDAO includes a liquidation mechanism that protects the protocol by auctioning NFTs when loans become under-collateralized, ensuring the system's stability. The BEND token is used for governance, enabling holders to participate in decision-making processes and influence protocol parameters.

6.8. Cyan

Cyan is a DeFi protocol offering Buy Now, Pay Later (BNPL) services and collateralized loans for NFTs, aiming to make NFTs more accessible. It provides financing solutions that allow users to acquire NFTs immediately and pay over an agreed period, or obtain loans against NFTs they already own. Cyan offers yield vaults where users can stake ETH or other assets to earn yields generated from lending activities, creating opportunities for passive income. The protocol employs valuation models to manage lending risks associated with NFTs, utilizing data-driven approaches to assess collateral value. By offering flexible financing options, Cyan lowers the entry barriers to NFT ownership and participation in the digital asset economy.

6.9. Airpuff

Airpuff is a DeFi protocol designed to optimize earnings from liquid restaking and restaking protocols, aiming to maximize user returns. It aggregates and enhances yields from restaking activities, allowing users to maintain liquidity while staking through liquid restaking mechanisms. Airpuff streamlines the process of participating in restaking, simplifying the user experience. The protocol implements strategies to reduce risks inherent in staking and restaking, such as smart contract risks and volatility. By focusing on yield optimization and risk mitigation, Airpuff seeks to provide users with improved returns on their staked assets.

6.10. Goldfinch

Goldfinch is a decentralized credit protocol offering crypto loans without requiring on-chain collateral, focusing on real-world borrowers and expanding access to capital in emerging markets. It facilitates unsecured crypto loans to businesses by leveraging off-chain collateral and rigorous borrower assessments. Goldfinch uses a network of auditors and backers to evaluate and approve borrowers, employing community-driven underwriting processes. Users, referred to as backers, can supply capital to fund these loans and earn yield, participating in the growth of global entrepreneurship. The protocol uses the GFI token for governance, allowing token holders to participate in decision-making processes and influence the protocol's direction.

6.11. Wildcat

Wildcat Protocol is a decentralized credit facilitation platform that enables pre-authorized borrowers to create custom lending markets with tailored parameters within protocol-defined limits. It allows vetted borrowers to set up markets with specific terms, offering personalized lending solutions while ensuring that market parameters remain within safe boundaries to protect lenders. Borrowers can curate a list of approved lenders for their markets, providing selective lender access and fostering trusted lending relationships. The protocol requires borrowers to undergo verification, such as Know Your Customer (KYC) processes, to maintain platform security and compliance. Wildcat aims to provide more personalized and flexible lending options within a decentralized framework, bridging the gap between traditional finance and DeFi.

7. Conclusion

In this paper, we introduced **De-FAULT**, a threat modeling framework specifically designed for DeFi lending protocols. Recognizing that traditional frameworks like STRIDE are insufficient for the unique challenges of decentralized finance—where protocols manage assets with intrinsic monetary value—we developed De-FAULT to address these complexities. The framework focuses on six critical categories: **Decentralized Issue**,

Coding Flaw, **Access Control**, **Upgradeable Contract**, **Business Logic**, and **Tampered Ratio**, providing a structured method for identifying vulnerabilities unique to DeFi lending.

We also established a set of essential invariants inspired by the Compound protocol's validation logic. These invariants serve as a checklist to ensure the robustness and integrity of protocol implementations. By applying the De-FAULT framework and these invariants to eleven different DeFi lending protocols—including Aave V3, Venus, and Euler V2—we demonstrated the framework's effectiveness in identifying potential vulnerabilities. Our analysis showed that protocols adhering closely to the invariants exhibited stronger security postures and were less susceptible to known attack vectors.

Our findings underscore the importance of adhering to fundamental security principles and the necessity for a specialized approach to threat modeling in DeFi applications. The De-FAULT framework offers a practical tool for developers and security professionals to proactively identify and mitigate potential threats during the development phase. By addressing the unique challenges of Web3 security, our framework contributes to strengthening the overall robustness of the DeFi ecosystem, fostering greater trust among users, and encouraging sustainable growth in decentralized financial services.