BlockApex Threat Modeling Service Report for Ember

Ember Threat Modeling

Process Applied

We have performed the threat analysis on the architecture of Ember protocol and built a model according to the aspects that are weak in architecture and should be changed. We did the analysis according to the STRIDE Model, ABC Threat Modeling and came to the conclusion that the protocol has too few improvements which are described below.

ASSETS

While Modeling the threats analysis on the Ember protocol, the following classes of business and data assets were identified. Furthermore, the stakeholders were also identified as threat actors or victims of threats.

Business Assets

Ember Renting Model

The renting protocol is probably the main business asset of the Ember protocol. It would be responsible for renting out the NFT and making sure that they are returned to the lender once the time and requirements are fulfilled.

Ember Lending Model

The lending mechanism is similar to renting protocol but the renting component and the component for lending out are separated.

Accounts/ Users

Users will include both the lenders and borrowers and the individuals who will connect their wallets with the protocol. This is also an asset because users are the one that brings business.

Underlying blockchain

This includes the blockchain on which the protocol is built, this is also an asset because the blockchain itself can be exploited as there many different possible attacks that are possible.

Ember Wallets

Wallets themselves are a separate component of the protocol and are different from individual users.

NFTs

NFTs are the main business asset of ember protocol. Because the lender can rent their NFTs into the ember protocol through which the lender can earn rent from the one who borrowed that NFT and later the borrower can use that rented NFT in games.

Data Assets

Encrypted Private Keys

This is one of the primary data that will be stored in the smart contracts, although the data will be double encrypted, regardless of it, the data itself is a valuable asset that will be used for the purpose of verification.

Encryption/ Decryption Scheme

The decryption mechanism and component are responsible for the verification of the borrower and making sure that the borrower is making the request to use the rented-out NFT.

Secret Keys

These secret keys will be in the encrypted format and under the possession of the borrower.

NFT Authorization Scheme

This component will be responsible for checking the authorizations on the rented NFT. It is a centralized component

Revenue Distribution Scheme

Finally this is another centralized component responsible for the distribution of rewards if the borrower/ lender has decided to pay the rent in terms of profit shares.

EOAs generating schema

On borrower renting request, the server generates an EOA, and NFT is transferred to that EOA. To generate EOAs the server uses a seed phrase to create the key pair and later that key pair is encrypted with the server's unique secret key and that encrypted private key is sent to the borrower.

Stakeholders/ Potential Threat Actors

Lender

Lenders would be lending out their NFT for renting. They would be the main stakeholders that will generate revenue by lending out their assets.

NFT Smart contract Owners

The lenders that would be lending out their NFTs for renting can be the owner of that NFT smart contract and that NFT owner could be a potential threat actor for the ember protocol because the owner of that NFT can execute all the onlyOwner functionalities in the NFT smart contract.

Borrower

Borrower will be renting the NFT that will be placed in the marketplace by lenders. They are one of the main stakeholders that could be a potential threat actor.

Custodial Component

- **EOAs** can be a threat to the lender because when the borrower rents the NFT it is placed into the wallet with the associated EOAs which could be a threat to the lender.

- **Decentralized Vaults**
  Smart contracts vaults themselves could be a threat to the lender as well as the borrower because after the NFT is rented it would be placed in the centralized vaults that could threaten themselves.

Rental Contracts

The decentralized smart contract that would be responsible for renting out the NFT and connecting the borrowers with lenders. They would be managing the requirements for each NFTs that are placed on the marketplace.

Participant Roles

There is a possibility that differences will be introduced in the protocol, that of the owner, governor or curator. This will eventually open further possibilities of more attack vectors.

Games/ Applications

Games are a major stakeholder because the rented NFT and assets will be used on top of them. Although currently it is not cleared in the team among what types of games or decentralized applications the protocol will be focusing on.

Centralized Infrastructure

Centralized infrastructure is playing a major part in threat analysis because the Ember wallets and decentralized vaults will be used through the centralized server, which is opening the system to countless different threats.

Farming Protocols Built On Top Of Ember

In the future, yield farming protocols might be built on top of the ember itself. This will open new attack and exploitation vectors to the protocol.

Community

Community itself is always a major stakeholder that plays a major part in the success of any product, especially in the case of the decentralized protocol.

Identified Threat Vectors In The Ember Architecture

By applying different threat models on the architecture of Ember, among the many different threats identified in the system we identified the following threats that are suggested to be addressed by Ember with immediate changes made to the system. All identified threat models are also provided in detail further below in this report.

Ineffectual authorization opens a window to server misuse

Problem Statement

The current architecture employs double encryption for users' private keys where the borrower or the centralized vaults never exercise full access to the EOA that holds the rented NFT during the lending borrowing cycle. First, the borrower needs to decrypt the double-encrypted private key with their own private key. After which, the semi-encrypted keys will be sent back to the server where the keys are further decrypted using the "secret key" residing on the server. The private keys are used once to create a transaction on behalf of the borrower, and finally discarded off the server.
The model described above is designed to introduce transparency and gain the trust of users for vaults, however, these vaults due to a set of requirements are centralized to some extent, covering up the internal flow or mechanism of the protocol as a whole.
The current model is over-engineered, missing the point to keep the private keys of EOA protected for all the stakeholders at all times. The architecture opens up a window to a set of vulnerable control flows, specific to the server, flagged during the threat modeling. Although the keys are not stored on the server permanently, there are multiple different ways through which the server is capable of gaining access to users' private keys e.g. monitoring the activity logs of the server, the presence of a malicious backdoor, etc. Therefore the mechanism for double encryption only brings latency issues and does not fulfill its purpose to the point.

Recommendation

Since there lie possibilities for the server to act malicious, to cost users' NFTs, ways for an adversarial backdoor or failing dependent library can still allow it to access keys though. The architecture should eliminate complexity by allowing the server access to private keys, in order to protect them. Alternatively, the keys should be saved in a secure and encrypted manner, completely eliminating the need for double encryption. The encryption and decryption mechanism could be implemented at the server layer and abstracted from the user for secure and increased usability with proper mapping maintained at the server side to keep track of all verified users.

Single Point-of-Failure

Problem Statement

Currently, due to the complex design of the server, there are multiple components that can act as a single point of failure rendering the protocol and ember wallet completely unusable. For example, the following attributes of the server if not handled properly or fails due to any reason will become a single-of-failure for the entire protocol.

- Encryption Decryption Mechanism
- Dependant libraries: Third-Party packages
- Directory traversals
- Misconfiguration attack
- Phishing attack
- Website defacement
- Unsecure network protocols usage

In case any of the above components becomes unresponsive or fails, the server can become victim to a Denial of Service attack. Hence, the problem lies with the system which currently lacks the robustness for such cases of DoS attacks making the protocol completely unavailable, creating a negative impact on the users' wallets.

Recommendation

It is recommended to set up a backup server as the whole information flow for the borrower depends on the server being online. The system downtime is unaffordable for this protocol because all the users' funds are directly at stake, for instance, all NFTs are stored in Ember Wallet during borrowing.

Price friction on upfront payable rentable NFTs

Problem Statement

The current system does not ensure prices of NFTs are updated as per the market. Any changes incorporated to the demand price of NFT, either off-chain or on-chain (through some bidding mechanism) are not reflected for the NFT which is currently taken up for rent with a fixed selling price. Although this demand of NFT is predictable, the price of NFT is immutable once listed for lending. In such a case, the lender will be able to unlend the NFT after the renting period is over and then list for lending again with increased rent charges as well as the sell price. But the issue lies in the MEV attack probability where the

call to unlend NFT after the renting period gets over can be frontrun with the buy call to sell the NFT at the previously fixed but now inappropriate price compared to the market demand.

Henceforth, we are confident to declare that the protocol is not able to tolerate price fluctuations for NFT during renting timeframes which could lead to a major loss for the lender who is forced to sell the NFT at a fixed listed price.

Recommendation

A mechanism is required to ensure the prices of all NFTs in the Ember ecosystem are reflected off the actual market value. The update price mechanism should be in place that allows the lender to update the previously listed NFT price to the actual indemand market value. Alternatively, an insurance mechanism to restrict lending the NFT at illegal prices is inevitable, that is, for NFT with a price varying from market price or the actual price derived from off-chain or on-chain pictures.

Ill-equipped mechanism for colluding scenarios

Problem Statement

An inherent threat that the system possesses is with the primary actors. The system is not steeled with mechanisms to prevent actors in the protocol from colluding with their authorized functionalities in order to corrupt the protocol. This issue is exploited as a possibility lies for collusion among the lenders, borrowers and on-chain gaming

protocols. In the current architecture there is no component to protect this kind of collision. Refer collusion matrix

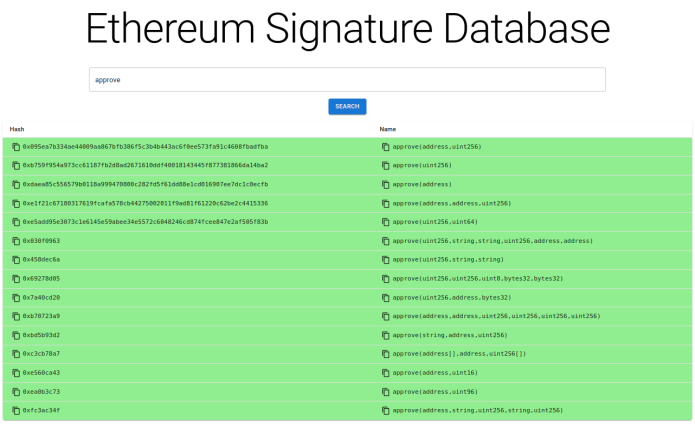
Recommendation

The mechanism to protect against the collision should be devised properly that would protect the business and data assets of ember.

Non-Standard NFTs bypassing server blacklisting

Problem Statement

According to the current architecture the design doesn't ensure that the NFT that has been listed into the protocol is according to the standards. Which opens the window for an attacker to bypass all the server blacklisting.
For example, a non-standarized NFT is listed into the protocol,performs the following functions with different signatures/methodID which doesn't adhere to the standard NFT smart contract pattern: For instance. From this source...



- Approve(Parameters)
- Transfer(Parameters)

Now the stopping mechanism implemented on the server site to stop any user from executing any sensitive functionality, checks the functions by comparing the function signatures send by the the contract with the signature that has been blacklisted on the server if it mathces then the server stops the execution or else it will allow the function to execute. In this way attacker can bypass this by requesting the sensitve functionalities with different signatures that perform the same actions that are restricted by borrower as mentioned above which will trick server into executing the blacklisted functionalites.

Recommendation

Ember protocol should implement a whitelisting policy which ensures that only the whitelisted signature will be used to perform certain funnctionalites.

Identified Threat Vectors - Developer Response

Identified Threat Vectors	Developer Response
Ineffectual authorization opens a window to server misuse	Acknowledged
Single Point-of-Failure	Acknowledged
Price friction on upfront payable rentable NFTs	Not fixed - (Reason -This issue is not fixed as the functionality is not currently a part of the protocol.)
Ill-equipped mechanism for colluding scenarios	Fixed
Non-Standard NFTs bypassing server blacklisting	Fixed

STRIDE In context of Blockchain

STRIDE was first devised by Microsoft to test traditional software applications but it is also mostly used for blockchain protocol to identify potential major issues in the protocol.

STRIDE: Threat Modeling

STRIDE Modelling for Ember Assets

The classification below shows the threat models that were identified on the stakeholders according to STRIDE.

	Spoofing	Tempering	Repudiation	Information Disclosure	Denial of Service	Elevated Privilege
Lender	Replaying signatures	-	-	-	-	-
Underlying NFT smart contract Owners	Replaying signatures	Can call burn function to burn all the NFTs rented by the users	-	-	Owners can perform functions to lock the NFTs forever in the vault	Owners can elevate privileges by executing all the Onlyowner functions in underlying smart contracts
Borrower	Replaying signatures	Social engineering attack against the borrower	Borrower can make the vault to transfer the NFT out of the vault	Leakage of encrypted private keys by the borrower	Losing access to the encrypted keys	Borrower is able to execute the transfer/ locking functions

Custodial Component / Decentralized Vaults	-	Backdoors	Use of malicious libraries on servers	Disclosure of private keys during verification Exposure of encryption mechanism Disclosure of private keys	NFT getting locked forever in the vault	System admin having access to vaults Attacker elevating the privilege after gaining access to the server
Rental Contracts	-	Plan modifications	-	-	Lender not able to receive the NFT after lending	-
Games/ Application	-	Unreliable and Unrealistic game requirement	-	-	Distribution of reward	-
Centralized Infrastructure	Attacker imitating the borrower	Admins making changes to the server	-	Attacker gaining access to the server	Server getting down	Backdoors placed by attacker, vulnerable libraries and untrustful team
Community	-	Community Losing trust on the protocol	-	-	-	-

Mitigation Matrix

The following matrix defines the mitigations for the threats identified above with the help of STRIDE classification, each threat is assigned an impact and likelihood and a suggestion to counter the threats presented. Countermeasures should be taken against each threat to make sure the protocol is free from any security loopholes.

Lender

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Replaying Signatures: Attackers able to replay the signatures Lender able to perform actions on NFT during the renting phase	Medium	Medium	Making sure that signatures cannot be replayed by adding a nonce and maintaining history NFT sub-letting should not allow the original lender to influence the decisions.	Enforce that original lender has no authority over the NFT during the renting.

Underlying NFT smart contract Owners

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Owners can execute Onlyowner functionalites from the smart contract and that functionalites can be malicious like locking the NFT forever into the vault or burning the NFTs	High	High	Make sure that the system doesn't allow any non-standarized NFT to be listed in the protocol.	Implement a proper whitelisting mechanism which ensures that no malicious NFT can be lended into the protocol.

Replaying Signatures: Attackers able to replay the signatures	Medium	Medium	Making sure that signatures cannot be replayed by adding a nonce and maintaining history	Enforce that original lender has no authority over the NFT during the renting.

Borrower

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Replaying Signatures: Signatures should not be replayed Borrower able to perform actions on the NFT even after the renting phase is over	High	Medium	As above a proper mechanism should be added to stop the attackers from replaying signatures as most of the functionalities would be performed through signatures.	Make a proper history of used signatures and maintain them to help tackle signature replay attacks.
Social Engineering: Attacker may manipulate user wallet, user interface or use social engineering trick user into signing transactions	High	Low	Educate users, team, protect the record and disable the caching on DNS server. Creating awareness about malicious activity among the borrowers.	Monitor the system, monitor the communities and impersonation of officials happening among the community of Ember, communicate the impersonations and phishing to the community, and have processes in place to quickly recover from

				attacks.
Borrower making the vault to transfer NFT out of the vault	High	Low	The custodial vault should properly blacklist the functions through which the NFT could be transferred out of the vault.	Blacklisting should be done properly so that no NFT could be transferred out of the vault.
Leakage of encrypted private keys by borrower	High	Medium	There is always the possibility of the borrower disclosing the double encrypted private keys or the semi encrypted private keys. If the semi-encrypted keys are disclosed then it could lead the attacker to execute functionalities in games and use that NFT.	Constant communication of the dangers of disclosing the private keys should be made to the borrower and how they could be exploited if the semi-encrypted private key is leaked.
Losing access to encrypted private keys	High	High	Borrowers could lose access to keys and the vaults currently don't have an emergency privilege to use in case the borrower lost the access. Adding a feature in the vault through which the server can give the	Server should be making a proper verifiable mapping of borrowers and which NFT they have borrowed. And use this mapping in case of emergency to give access to keys to the wallet again.

			borrower access back to the borrower if the keys are lost.	
Borrower is able to execute the transfer/ locking functions	High	Low	Make sure that the vaults are fully protected from any malicious behaviors through which the NFTs could be locked forever.	Regularly perform the audit of the code base and take help from auditing firms to help tackle these issues.

Custodial Component (EOAs, Decentralized Vault)

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Backdoor placed in custodial vaults which is created on a server	High	Low	Try to keep the custodial vaults as decentralized as possible but the vault will be centralized due to which there will always be a chance of centralized influence on it. Making sure that centralized and decentralized components are separated from each other and the server doesn't have influence on vaults. So in case of	Separating the influence of components would protect the vaults from backdoors on the server. Make sure that in future updates to the server, it is not able to influence the vaults.

			backdoor there is much the malicious actor is able to do	
Usage of malicious libraries: servers on which the vaults are implemented could be using vulnerable libraries	Medium	Medium	The code should be less dependable on too many different third party softwares.	Each third party software should be updated regularly with schedule to be sure they are not vulnerable
Disclosure of private keys during the encryption and verification process	High	High	Although the private keys are themselves not stored on the server, there would be a point in time during the decryption and verification phase when the private keys will be visible to the server and backend. Due to which this information will be stored in the server logs and it would be visible to any malicious actor if that actor is present on the server.	Sensitive information should not be logged, and the decrypted private keys should only be stored in temporary memory instead of permanent memory.
NFT getting locked forever in the vault due to a DoS and vaults having no authority is case of	High	High	During the renting phase the vault currently will have no authority over the NFT other	As explained in the previous threat, the server will have private access to private despite the keys not getting saved, there should

emergency			than storing them in an EOA. Incase of technical issue or borrower losing access to keys the NFT would be locked in the vault. This would also lead to other issues, if due to some technical flaw the NFT is rented for an indefinite period then there is a higher chance for permanent DoS on the NFT and it could be lost forever.	be a component for emergency exit in case of DoS. Although this component should be carefully planned so that other malicious actors will not be able to get access to sensitive keys.
System admin having access to the vaults	High	Low	Access privilege of system admin should be limited.	System admins should not be given full privilege and access to each component.

Rental Contracts

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Plan Modification: Modifications of plan through addition of other modules in the protocol	Medium	Low	Make sure the dev team of Ember is able to make drastic changes to the protocol through ownerships or governance and each	Implement and create a DAO to keep the community engaged with the changes.

			changes should be done through the use of DAO	
Lender not able to receive NFT after lending: Due to a DoS flaw in the contract or collision of requirements the NFT gets locked in the protocol forever	High	Medium	Develop the contract by following secure design patterns. Have the contract properly audited from at least 2 different firms to be sure that it is free from any flaws.	After each new contract is developed, alway have it audited from third party firms and experts to protect it from unintended behavior.

Games/ Application

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Unreliable and Unrealistic requirements of games: Collusion among the game requirements and lending requirements which would make the rented NFT unusable	Medium	Low	There is no proper mechanism to protect from this business issue. Make sure that the requirements that the lenders will set are feasible with the games currently available. Or else this could lead to business stagnation.	A range of requirements should be made available to the lender that are according to the standards of games and applications.

Distributions of reward by games: Although this would not directly affect the protocol as this depends on this party games but could affect the NFT rental model and indirectly affect the usability of Ember	Medium	High	Eventually the borrower will use the NFT in games. Make sure the games available are profitable with the business model of Ember.	Design and keep updating the business model according to the games available.

Centralized Infrastructure

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Attacker Imitating the borrower: As the custodial vault will be on a centralized server there is high probability of bypassing the protection mechanism and attackers imitating the borrower if they are simply able to get the encrypted keys	High	High	Our recommendation would include to use as less centralized components as possible because of the possibility of terpering the records. On the server side proper authentication and authorization behind each functionality. Make sure that the wallet that has borrowed the NFT only	Usage of encryption mechanisms would be helpful.

			that wallet is able to make transactions and make the server to execute functionalities.	
Admins making changes to server: The infrastructure will be under the full control of team, due to which the any member of the team could be exploiting the privilege	High	Low	There should be more than one single authority that would be allowed to make changes to the server. Make more than one server admin and whenever a change is required make sure that all the admin have granted the permission to make changes.	Admins should be restricted to execute sensitive functionalities and make changes to the backend components. Admin should not be granted full privilege.
Attacker gaining access to the server: Whenever a high profile hack happens on centralized infrastructure the attacker gains full admin access to the server.	High	High	The major drawback of using a centralized system in a decentralized protocol is higher chances of getting hacked and an attacker getting all the secret keys and access to logs. The use of logs should be omitted and	Don't store sensitive data in backup files and server logs.

			make sure that no information is stored in the server logs and backup files that could cause harm to the protocol in case of the server getting compromised.	
Server getting down: Through a DDoS attack or other technical failure. This will be a single point of failure for the whole protocol	High	High	This is a single point of failure in the protocol, and in case when the server will get down the renting feature of the protocol will be unavailable and NFT will be locked for the time being. Add a backup server that will be activated in the event of main server getting down due to technical problem and DDoS	Adding a backup server resource will be the only good countermeasure to save the protocol from DoS.
Backdoor places by attackers, outdated libraries & dependencies, vulnerable server versions and 0days	High	High	The best measure to protect from 0day would be to update the libraries and frameworks to the latest versions and making sure that no	Make a schedule to regularly update the libraries and dependencies to make sure that server is fully protected from 0day vulnerabilities.

			vulnerable version or component is used on the server.	

Community

Threat Vector	Impact	Likelihood	Mitigation	CounterMeasures
Community Losing trust on the protocol	High	Unidentified	Stay connected with the community. Keep evolving the protocol according the community needs	Monitor the responses and behavior of community about each component of the Ember protocol

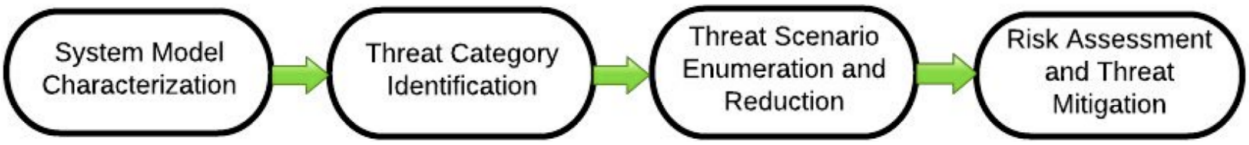
Developer Response:

All risks have been addressed

ABC: A Cryptocurrency-Focused Threat Modeling Framework

- A systematic threat modeling framework geared toward cryptocurrency-based systems.
  - Its tools are useful for any distributed system.
- Helps designers to focus on:
  - Financial motivation of attackers.
  - New asset types in cryptocurrencies.
  - Deriving system-specific threat categories.
  - Spotting collusion and managing the complexity of the threat space.
    - Using a new tool called a collusion matrix.
- Integrates with other steps of a system design; risk management and threat mitigation

Abc Steps



Ember Project Description

Ember is the first sharing economy for NFTs, gaming, finance & entertainment. Ember is automatically compatible with nearly every game & NFT asset, introducing a new own-to-earn direction for Web3.

Ember is composed of Ember Marketplace and the Ember NFT Wallet, both of which are powered by the Ember Renting Protocol, allowing everyone to rent, borrow, and share digital assets.

Step 1: System Model Characterization

Identify the following:

- Activities in the system.
- Participant roles.
- Assets.
- Any external dependencies on other services.
- System assumptions.

- Draw a network diagram(s) of the system modules

Activities In the System

- Lending/ Renting
  - Direct Lending
    - customized especially for guilds to lend NFTs on yield to their best Scholars
  - Yield Distribution
  - Bulk Lending
  - Sub-Lending & Multi-Renting
- Borrowing
  - Bulk Borrowing

Participant Roles

- Lender
- Renter
- Borrower

ASSETS

Business Assets

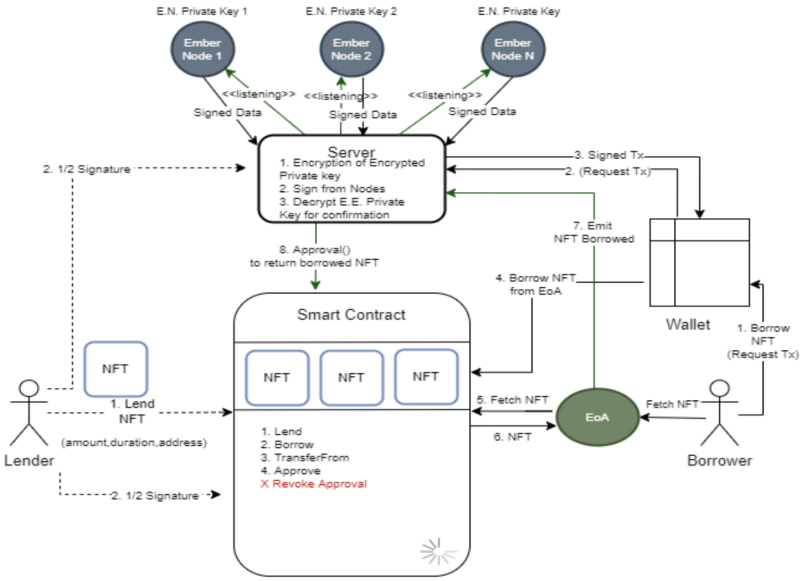
- Ember Renting Model
- Accounts/ Users (EOA Wallets)
- Underlying blockchain
- Lending Model
- Ember Wallets
- Underlying NFT

Data Assets

- Encrypted Private Keys
- Encryption/ Decryption Mechanism
- Secret Keys
- NFT Authorization schema
- Revenue distribution schema
- EOAs generating schema

System Assumptions:
This system assumption requires the borrower to make an up-front payment that enables uncollateralized NFT rentals. The rented NFT is escrowed in a registry smart contract which defines a set of access constraints for the renter.

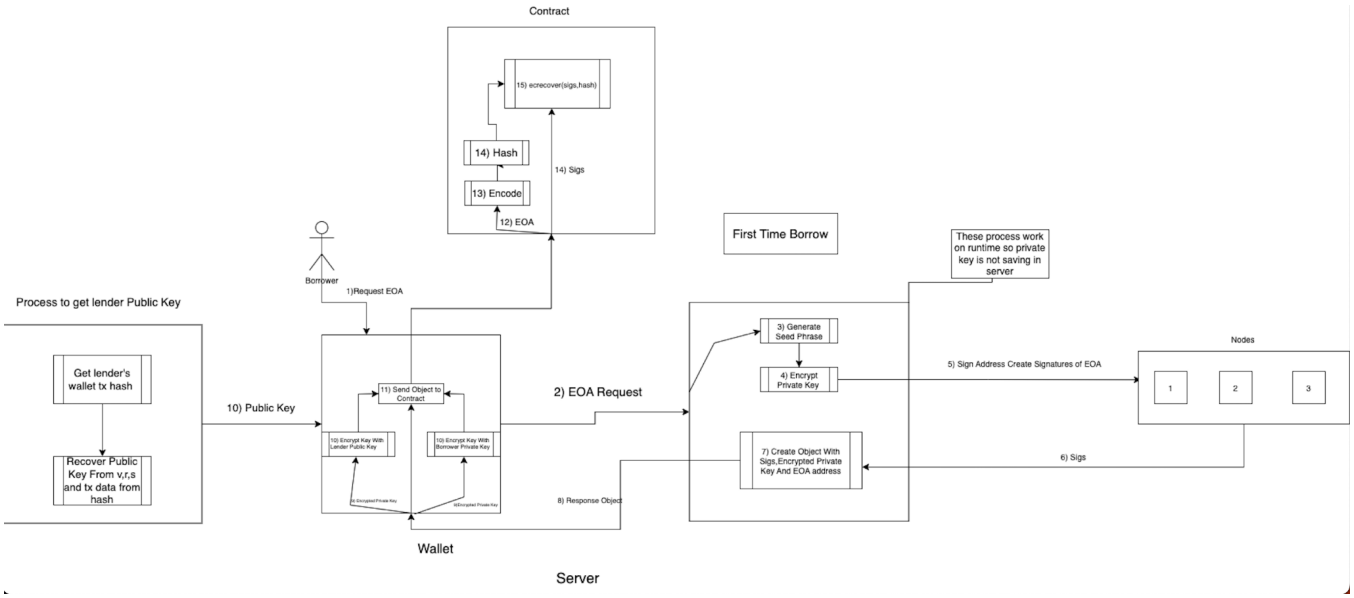
Ember Network Diagram
Functionality Description: allowing everyone to lend, rent, and borrow digital assets (NFT's)
Participants: Lender, Borrower
Assets: Server, Encrypted Private Keys, Encryption/ Decryption Mechanism, Secret Keys Authorization to execute the functionalities on NFT, Distribution mechanism of Revenue shares



Authorization Ember user data flow diagram

Functionality Description: A borrower can borrow or rent an NFT to be used with third party gaming applications.

Participant: Borrower



Step 2: Threat Category Identification

- ABC defines broad threat classes that must be investigated.
- ABC defines these classes around the assets.
- For each asset:
  - Define what constitutes a secure behavior for the asset.
  - Use that knowledge to derive the asset security requirements.
  - Define threat classes as violations of these requirements.

Data Assets	Security Threat Category
Server	**Denial of Service** (Server getting down can result in a loss of users NFTs) **Using Component with known vulnerabilities** (Server using vulnerable libraries and components can lead to the loss of access to the server) **Signature Replay Attack** (Server is able to replay a signature through the signature verification process) **Social Engineering** (Server credentials are maliciously utilized)
Secret Keys	**Insecure storage of Secret Keys/ Encrypted Private Keys** (Users' private keys can be stolen incase the server is compromised) **Social Engineering** (Lender/ Borrower private keys/ Encrypted private keys/ NFTs are lost due to malicious social engineering activities like phishing etc)
Encryption/Decryption Mechanism	**Inefficient encrypting and decrypting methods** (Inadequate secret key encryption methods can be brute forced to reveal and further repudiate sensitive data)
NFT Authorization schema	**Maliciously executing unauthorized functionalities on NFT** (Abusing system's integrity by executing unauthorized functionalities for an NFT)

Revene Distributions schema	**Inappropriate distribution of revenue** (Manipulation in server properties which are responsible for adequate distribution of shares)
EOAs generating schema	**Inefficeint way of generating EOAs** (inadequate way of generating EOA's can be brute forced and leads to attacker controlled EOAs)

Step 3: Threat Scenario Enumeration and Reduction

- For each threat, define scenarios that attackers may follow to pursue their goals.
    - It is comprehensive, and considers collusion and financial motivation.
- ABC devises collusion matrices to help with this step.
- Analyzing a collusion matrix involves:
    - Enumerating all possible attack scenarios.
    - Cross out irrelevant cases and merge together those that have the same effect.
    - Documenting all distilled threat scenarios.

Threat 1: Servers

Denial of service

Attack Scenario: Servers crashing is an unexceptional event in the software industry with a number of legitimate or malicious reasons. For example, power failure, force majeure, hardware failure, or a simple human error are considered as valid faults but the most common of the server crashes are due to malicious activities like DOS or DDoS attacks.

Denial of Service (DoS) occurs when the network gets flooded with traffic ultimately disrupting the user and forbidding them access to the data. The hacker overloads the server with traffic by sending legit but recurring commands causing it to crash. Distributed Denial of Service (DDoS) is the same as a DoS attack but comes from a vast number of machines working towards the same goal. Since there's an increased attack surface, a DDoS attack is more sophisticated and harder to protect against.

In any case, if the server crashes all of its actions are stopped, creating a huge impact on the users of the Ember protocol as the server currently is acting like a centralized authority and almost all the operations are being performed through it. Server crashing can create chaos among the users and indeed a loss of trust for the protocol.

Using Components with known vulnerabilities

Attack Scenario: Servers are made up of different components and packages. The components can be vulnerable, given a dependent library fails due to a new bug. So If the server's code containing a vulnerable version of the library doesn't update on time or is

patched, the result is a weakness for the protocol. Attackers can view the versions of the dependencies containing the vulnerable code and confirm the exploit as easily as by checking it on websites such as exploit-db.

Due to the large complexity of modern applications, it is easy to lose sight of all the dependencies and software being used, commonly termed as a SBoM. It is important to know that automated scanners or manual testing might reveal outdated software with issues. Exploiting known issues can have disastrous impacts, they are often the first thing the attackers look at and abuse to gain a foothold, elevate their privileges, impersonate other users and whatnot. If the attacker gets hold of the server all the users can lose their funds and NFTs.

Signature Replaying Attack

Attack Scenario: Server can execute the same signature again if a proper signature verification mechanism is not implemented through the system complex, which includes the Server primarily and is supported by the underlying Smart contract. Using the same signature can lead to the execution of the offered functionalities more than one time which originally requires unique signatures.

Social Engineering Attack

Attack Scenario: Nowadays it's common to host the server on cloud infrastructure such as AWS so in that case if the attacker gets the admin console credentials of the server through social engineering then it can have full control over the server and can execute malicious activity which will create a financial loss for the users of ember protocol.

Threat 2: Secret Keys

Insecure storage of Secret Keys/Encrypted private key

Attack Scenario: According to the architectural flow of the server, on a lending transaction, a request is sent to the server from the borrower's wallet. The server generates a keypair and after encrypting the private key from its secret key the private encrypted key is sent to the borrower and then the server loses the keys. Note that there is a time during this flow in which the key pair and the encrypted private keys remains on the server before sending it to the borrower so if the server is compromised or the keys are intercepted then the attacker can easily get hold of the private keys which leads to the loss of users' NFT and funds.

Social Engineering Attack

Attack Scenario: Social engineering attacks refer to a range of malicious activities where bad actors trick users into making security mistakes like signing transactions or giving away sensitive information. Attackers use impersonation techniques, psychological

manipulation, and falsified human interactions to gain the trust of targets before making off with their crypto holdings, NFTs, private keys, or other such assets.

Threat 3: Encryption/Decryption Mechanism

Insecure way of encrypting and decrypting data

Attack Scenario: As the design flow states that upon lending, the borrower will send a request to the server, the server will generate a keypair using a seed phrase. After which, the server will encrypt the private key with the server's secret key. Now the server's secret key here raises security concerns, if the server is using a weak secret key to encrypt the private key then an attacker can easily brute force that secret key. Which gives attackers full authority to decrypt and encrypt private transactions, and can easily hijack users' funds and NFTs

Threat 4: Authorization to execute the functionalities on NFT

Maliciously executing unauthorized functionalities on NFT

Attack Scenario: The server plays a very important role in the authorization of all the actions that the borrower can perform. To give some perspective the borrower can not execute actions using the borrowed NFT which the borrower is not authorized to or the lender doesn't allow. For example, borrowers can not execute transfer functionality on the NFT contract or can not interact with the NFT contract directly, every request has to go through the server. So we can say that the server here is acting like a central authority. The security concern here is that the server can be compromised or the server request can be tempered using a non standarized NFT which leads to the attacker/borrower performing an unauthorized action on the server which the borrower is not allowed to do. This as a result gives a financial loss to the lender because his NFT can be compromised. We have seen a lot of attacks like this in web2 in which whole systems are compromised just because of unprotected or unsecured servers.

Threat 5: Distribution mechanism of Revenue shares

Inappropriate distribution of revenue

Attack Scenario: Rewards are distributed among users through servers so in that case, a malicious user can trick the server to send inappropriate distribution of revenues among users. It is possible because if the formula used by the smart contract is miscalculating the revenues or the server request has been tempered by the attacker then the distribution of rewards can be at risk.

Threat 6: EOAs generating mechanism

Inefficeint way of generating EOAs

Attack Scenario: The server generates EOAs on renting request by borrower on which the NFT is transferred and later that EOA's are used to perform the functionalities on the NFT. so if the EOA's generating mechanism is not upto the mark and could easily be brute forced then the attacker can generate their own EOA's and execute malicious functionalites through the server.

Collusion Matrix

Service Theft Threat Collusion Matrix

Targets

Attackers	lender	Borrower	Server	Borrower and Server	Lender and Server
Lender	- -	- -	Signature Replaying attack, the lender can use the same signature more than one times on the server to lend/rent the NFT in the vault.	- -	- -
Server	Lenders NFT getting locked in the vault due to the server crashing. Lenders' NFTs get stolen due to the loss of secret keys from the server	The borrower is not able to return the NFT or to pay rent due to the server crashing In Compromised server the attacker can get hold of the private keys of borrowers executing unauthorized transfer	Server admin credentials get leaked via social engineering attack which results in the attacker gaining complete access over the server.	- -	- -

		functions.			
Borrower	not returning lenders NFT after duration expired Borrower not paying rent.	Borrower private keys get leaks due to social engineering	Malicious borrower tricking Server into performing inappropriate revenue distribution. .	- -	Borrower not returning the NFT to the lender through Server. Malicious borrower pays less or no amount of rent to lenders by tempering the request on the server during renting process
Borrower and server	Malicious borrowers can execute unauthorized functionality with the help of compromised servers like transferring the ownership from lender to borrower.	- -	- -	- -	- -

Step 4: Risk Management and Threat Mitigation

● An independent task of threat modeling.

Threat Mitigations:

- Make sure to implement proper security measures on the server which ensures that the attacker gets a hard time compromising the server. A strong password should be used for the authentication of the server. SSH should be securely implemented which doesn't allow the attacker to connect to your server directly.

- Implement a Web application firewall (WAFF) like Cloudflare on the server to stop application-level vulnerabilities.

- Make sure that the server has properly implemented rate limiting. Through this, the server can limit the malicious traffic coming from users by blacklisting or blocking the IPs permanently or temporarily. Load Balancers should be in place if in case of server crashed

- Make sure to use a strong secret key generating mechanism that ensures that the secret key can not be easily brute-forced by the attacker.

- Make sure the server doesn't use any outdated libraries or packages which have a security vulnerability. Implement a mechanism to check outdated dependencies because 0-day vulnerabilities can arise anytime.

- Give awareness to your protocol users about social engineering attacks and how easily attackers can steal valuable pieces of information like private keys.

Developer Response:

All risks have been addressed. However, there is currently no usage of WAFF on the server.