# SmartCredit Security Analysis

## by Pessimistic

This report is public

January 21, 2022

# Abstract

In this report, we consider the security of smarts contracts of SmartCredit project. Our task is to find and describe security issues in the smart contracts of the platform.

# Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

# Summary

In this report, we considered the security of SmartCredit smart contracts. We performed our audit according to the procedure described below.

The initial audit showed a critical issue Locked tokens and many issues of medium severity including Project design, Inoperable contract, Wrong balance check, Access control, and others. Also, several issues of low severity were found. The tests fail to run.

After the initial audit, the code base was updated. The majority of the issues were fixed, however new issues (including one critical bug and two medium severity issues) were discovered.

Later the code base was updated again. In this update the bug and some other new issues were fixed. The test issue was as well resolved. No new issues were found.

# General recommendations

We recommend adressding the remaining issues. We also recommend adding CI to analyze code with linters and security tools.

# Project overview

## Project description

For the audit, we were provided with SmartCredit project on a private repository, commit ad35039807a396ae89fcd36eea0c0881fdb8af59.

The project has a documentation and README.md file. However, the code base lacks NatSpec comments.

Tests are outdated and fail to run.

The total LOC of audited sources is 1347.

## Code base update

After the initial audit, the code base was updated. For the recheck, we were provided with commit d2e836989f224f14905bf05b417dcfa2f34b1fce.

In this update the majority of issues were fixed, total LOC increased to 1727, the code base was refactored. With this version (1.2) the following major changes were made:

- A new **NFTLoan** (with ERC721 as collateral) contract was introduced to the SmartCredit ecosystem.
- Support of SmartCredit token was added to fixed income funds with new staking functionality.
- A loan request with EIP712 signature was implemented (see `CreditLine.initializeLoan`).
- The **CreditLine** has been implemented in four different versions.
- The liquidation mechanic was recomposed via a new **Liquidator** contract.

All the changes were provided with the corresponding documentation. All of them were audited for security issues.

## Code base update #2

After the first recheck, the code base was updated again. We were provided with commit 924752d653a58972edb673dbc53c0ba0484c19eb.

This update includes fixes for the **NFTLoan** contract and for the tests of the project. No new functions added, no new issues found.

In this update 170 out of 171 tests pass, the code coverage is 94,56%.

# Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
    - We scan project's code base with automated tools: Slither and SmartCheck.
    - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
    - We manually analyze code base for security vulnerabilities.
    - We assess overall project structure and quality.
- Report
    - We reflect all the gathered information in the report.

# Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

## Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

### Locked tokens (fixed)

`fixedIncomeFundBalance()` function of **ERC20CreditFIF** contract is used to get the amount of tokens on the contract. However, the function returns Ether balance of the contract. Therefore, all the checks that require a certain amount of tokens to present on the contract will revert if the contract does not have the same amount of Ether on its balance. Since the contract does not accept Ether, the function will always return `0`. Thus, tokens will be stuck on the contract.

*The issue has been fixed and is not present in the latest version of the code.*

### Invalid lender value (fixed)

In **NFTLoan** contract there is a bug in `acceptLoanRequest` function: it receives user's tokens, but does not initialize the `lender` field of the loan with user's address. Hence, on the repay stage, the contract will transfer repaying tokens to zero address (line 131). Another scenario (claiming collateral in case of unpaid credit) will revert, as it requires `msg.sender` to be zero (line 159). We recommend running tests for such cases.

*The issue has been fixed and is not present in the latest version of the code.*

## Medium severity issues

Medium issues can influence project operation in current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

### Access control (fixed)

Function `investFixedIncomeFundToCompound()` of **SmartCredit** contract allows a user with `CREATOR_ROLE` role to call any function of any contract on behalf of **SmartCredit** contract and thus bypass the check of `onlySmartCredit` modifier. As a result, a user with this role can perform actions that are not supposed for this role.

We recommend explicitly specifying the call to `investToCompound()` function inside `investFixedIncomeFundToCompound()`.

*The issue has been fixed and is not present in the latest version of the code.*

### Inoperable contract (fixed)

**ETHFIF** contract is intended to work with Ether transactions. However, the contract is unable to accept Ether since it does not have `receive()` function or `payable` fallback function.

*The issue has been fixed and is not present in the latest version of the code.*

### Wrong balance check (fixed)

In **CreditLine** contract, `repay()` and `finishLiquidation()` functions are used to pay off a debt in ERC20 tokens. However, these functions check the balance in Ether, not in ERC20 tokens. Thus, the user has to have at least 1 Ether per ERC20 token to perform a transfer.

*The issue has been fixed and is not present in the latest version of the code.*

### Forged contracts (fixed)

Function `createFixedIncomeFund()` of **SmartCredit** contract allows to create new funds with arbitrary `_fundData` parameter. As a result, a malicious user can create a new contract that belongs to another user but has modified `smartCreditRegistry` address. This allows an attacker to verify transactions on behalf of SmartCredit and thus to steal users' funds. This can potentially be used to trick users and decrease the trust level.

We recommend performing on-chain validation of the `smartCreditRegistry` address when initializing new fund.

*The issue has been fixed and is not present in the latest version of the code.*

### ERC20 standard violation (fixed)

EIP-20 states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, in `withdraw()` function of **USDLiquidator**, the returned value of `transfer()` call is not checked at line 70.

*The issue has been fixed and is not present in the latest version of the code.*

### Tests issues (fixed)

The project has tests. However, the tests are outdated and fail to run.

Testing is crucial for code security and audit does not replace tests in any way. We highly recommend covering the code with tests and making sure that the code coverage is sufficient.

*The issues have been fixed and are not present in the latest version of the code.*

### Overpowered role

**SmartCredit** contract heavily relies on Registry: all intercontract coordination is implemented as calls to addresses obtained from Registry, moreover roles access control is designed relying on it. Since **SmartRegistry** is an ownable contract, in case of compromising owner's key, an attacker can reassign the admin role in the whole system to themself and change the addresses of the SmartCredit subsystems (`setAddress`) for malicious contracts.

Thus, the loss of control over **SmartRegistry** results in loss of control over the main contract (it cannot be paused when the admin role is changed). Consider implementing multisig for changing crucial protocol parameters (i.e. subsystem addresses and roles), and/or making the emergency pauser role (admin) independent of other roles.

### Locked ether

**SmartCredit** contracts implements the `receive` function for accepting ether, but does not have a function to spend it. Thus, all the ether send to contract would be locked. Consider implementing a function to withdraw sent ehter or removing receiving function.

### Project design

The overall project architecture is consistent. However, certain parts of the code are copy-pasted without proper refactoring, e.g. **ERC20CreditFIF** and **ETHCreditFIF** contracts. This decreased the maintainability of the code base, resulted in several vulnerabilities, and made the whole project almost inoperable.

We recommend refactoring the code base to avoid code duplication.

In the updated code modularity has increased. Some functions were refactored to more clear abstraction layers, but some (i.e. **StakingFIF**) were not. Due to increased complexity, we still recommend paying extra attention to architecture issues.

## Low severity issues

Low severity issues do not directly affect project's operations. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

### EIP721 safe transfer not supported (fixed)

Note, that the standard includes safe transfers feature, which is currently not supported by **NFTLoan** contract. Consider implementing `ERC721TokenReceiver` interface to enable such support.

*The issue has been fixed and is not present in the latest version of the code.*

### Code quality

- The code base almost lacks NatSpec comments. Consider adding them to improve code maintainability.

  *The issue has been fixed and is not present in the latest version of the code.*

- In **CreditLine** contract, events declared at lines 55–60 are never emitted.

  *The issue has been fixed and is not present in the latest version of the code.*

- Consider using `address => bool` mapping instead of `creditLineAddresses` array to simplify the code and optimize gas consumption in **ETHCreditFIF** and **ERC20CreditFIF** contracts.

  *The issue has been fixed and is not present in the latest version of the code.*

- In **SmartCredit** contract at line 64, `borrower` parameter of `CreditLineCreated` event should be `indexed`.

  *The issue has been fixed and is not present in the latest version of the code.*

- Hadrcoded address at **LoanContractTypeE** at line 12.

- Consider rewriting the expression of type

  ```
  if (condition) return false;
  else return true;
  ```

  as `return !condition;` to improve code readability in **CreditFIF**, **Liquidator** and **StakingFIF**.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

January 21, 2022