



# SMART CONTRACT SECURITY AUDIT OF **VELAS DOMAIN**



**SMART CONTRACT AUDIT | SOLIDITY DEVELOPMENT & TESTING | PROJECT EVALUATION**

**RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN**

# Audit Introduction

<b>Auditing Firm</b>	InterFi Network
<b>Audit Architecture</b>	InterFi Echelon Auditing Standard
<b>Language</b>	Solidity
<b>Client Firm</b>	Velas Domain
<b>Twitter</b>	<a href="https://twitter.com/vlxdomains/">https://twitter.com/vlxdomains/</a>
<b>Report Date</b>	August 16, 2022

## Contract addresses:

InterFi

BaseRegistrarImplementation: 0x40C9Eeae07f038DdD9b987846dEaD1483a7C71aD

ETHRegistrarController: 0x9413E349cAa08fa2e639c99c27091479b7e0481D

StablePriceOracle: 0xFce805fAe2e5Deb6fC141C929795becc7d12a566

ReverseRegistrar: 0x206340c328E88A45fC33BEf42129fafE9Ff9f980

DefaultReverseResolver: 0x4688776634373C970659355Cee2616074D882660

PublicResolver: 0x1C332E63f266c27F7f55aF3dF8f3502603362cAB

## Blockchain:

Velas Chain <https://evmexplorer.velas.com/>

✅ Verify the authenticity of this report on InterFi's GitHub: <https://github.com/interfinetwork>



# Audit Summary

InterFi team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- ❖ Velas Domain's solidity source codes have **LOW RISK SEVERITY**
- ❖ Velas Domain's smart contracts have **ACTIVE OWNERSHIP**
- ❖ Velas Domain's centralization risk correlated to the active owner is **HIGH**
- ❖ Important contract privileges – **ADD REMOVE CONTROLLER, SET PRICE ORACLE, SET PRICES, SET COMMITMENT AGES, SET NAME**

Be aware that smart contracts deployed on the blockchain aren't resistant to internal exploit, external vulnerability, or hack. The scope of this audit treats out-of-scope entities as black boxes and assumes their functional correctness. However, in the real world, out-of-scope contracts can be compromised, and exploited. Moreover, changes in out-of-scope contracts can create severe impacts. For a detailed understanding of risk severity, source code vulnerability, exploitability, and audit disclaimer, kindly refer to the audit.



# Table Of Contents

## **Audit Information**

Audit Scope .....	5
-------------------	---

## **Echelon Audit Standard**

Audit Methodology .....	6
Risk Classification .....	8
Centralization Risk .....	9

## **Smart Contract Risk Assessment**

Static Analysis .....	10
Software Analysis .....	14
Manual Analysis .....	16
SWC Attacks .....	19
Risk Status & Radar Chart .....	21

## **Audit Summary**

Auditor's Verdict .....	22
-------------------------	----

## **Legal Advisory**

Important Disclaimer .....	23
About InterFi Network .....	26



# Audit Scope

InterFi was consulted by Velas Domain to conduct the smart contract security audit of their solidity source codes. The audit scope of work is strictly limited to the mentioned solidity file(s) only:

- ❖ BaseRegistrar.sol
- ❖ BaseRegistrarImplementation.sol
- ❖ BulkRenewal.sol
- ❖ ETHRegistrarController.sol
- ❖ PriceOracle.sol
- ❖ StablePriceOracle.sol
- ❖ ReverseRegistrar.sol
- ❖ TestRegistrar.sol
- ❖ DefaultReverseResolver.sol
- ❖ ISupportsInterface.sol
- ❖ Multicallable.sol
- ❖ PublicResolver.sol
- ❖ Resolver.sol
- ❖ SupportsInterface.sol
- ❖ INameWrapper.sol
- ❖ ERC1155Fuse.sol

InterFi

Smart Contract  
Security Audit

## **Audit Hash**

Solidity source code is audited at hash #35be05ca1a080b5021597609f23130bfa746ea0d



# Audit Methodology

The scope of this report is to audit smart contract sources code of Velas Domain. InterFi has scanned contracts and reviewed codes for common vulnerabilities, exploits, hacks, and back-doors. Due to being out of scope, InterFi has not tested contracts on testnet to assess any functional flaws. Smart contract audits are conducted using a set of standards and procedures. A mutual collaboration is essential to perform an effective smart contract audit. Here's a brief overview of auditing process and methodology:

## **Connect**

- ❖ Onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

## **Audit**

- ❖ Automated analysis is performed to identify common contract vulnerabilities. We may use following third party frameworks and dependencies to perform the automated analysis:
  - Remix IDE Developer Tool
  - Open Zeppelin Code Analyzer
  - SWC Vulnerabilities Registry
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- ❖ Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- ❖ Manual line by line analysis is performed to identify contract issues and centralized privileges.



## **Report**

- ❖ Auditing team provides a preliminary report specifying all the checks which have been performed and findings thereof.
- ❖ Client's development team reviews the report, and makes amendments in solidity codes.
- ❖ Auditing team provides the final comprehensive report with open and unresolved issues.

## **Publish**

- ❖ Client may use the audit report internally or disclose it publicly.
- ❖ It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

## **Automated 3P frameworks used to assess the smart contract vulnerabilities**

- ❖ Remix IDE Developer Tool
- ❖ Open Zeppelin Code Analyzer
- ❖ SWC Vulnerabilities Registry
- ❖ DEX Dependencies, e.g., Pancakeswap, Uniswap



# Risk Classification

Smart contracts are generally designed to manipulate and hold funds denominated in ETH/BNB. This makes them very tempting attack targets, as a successful attack may allow the attacker to directly steal funds from the contract. Below are the typical risk levels of a smart contract:

**Vulnerable:** A contract is vulnerable if it has been flagged by a static analysis tool as such. As we will see later, this means that some contracts may be vulnerable because of a false positive.

**Exploitable:** A contract is exploitable if it is vulnerable and the vulnerability could be exploited by an external attacker. For example, if the “vulnerability” flagged by a tool is in a function that requires owning the contract, it would be vulnerable but not exploitable.

**Exploited:** A contract is exploited if it received a transaction on the main network which triggered one of its vulnerabilities. Therefore, a contract can be vulnerable or even exploitable without having been exploited.

Risk severity	Meaning
<b>! High</b>	This level vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
<b>! Medium</b>	This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
<b>! Low</b>	This level vulnerabilities should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy related vulnerabilities should be fixed to deter exploits.
<b>! Informational</b>	This level vulnerabilities may be ignored. They are code style violations and deviations from standard practises.





# Centralization Risk

Centralization risk is the most common cause of decentralized finance hacks. When a smart contract has an active contract ownership, the risk related to centralization is elevated. There are some well-intended reasons to be an active contract owner, such as:

- ❖ Contract owner can be granted the power to `pause()` or `lock()` the contract in case of an external attack.
- ❖ Contract owner can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale, and to list on an exchange.

Authorizing a full centralized power to a single body can be dangerous. Unfortunately, centralization related risks are higher than common smart contract vulnerabilities. Centralization of ownership creates a risk of rug pull scams, where owners cash out tokens in such quantities that they become valueless. **Most important question to ask here is, how to mitigate centralization risk?** Here's InterFi's recommendation to lower the risks related to centralization hacks:

- ❖ Smart contract owner's private key must be carefully secured to avoid any potential hack.
- ❖ Smart contract ownership should be shared by multi-signature (multi-sig) wallets.
- ❖ Smart contract ownership can be locked in a contract, user voting, or community DAO can be introduced to unlock the ownership.

## Velas Domain's Centralization Status









- ❖ Velas Domain's smart contracts have **active ownership**.
- ❖ Smart contracts are **not deployed** on blockchain at the time of the audit.



# Static Analysis

Symbol	Meaning
	Function can modify state
	Function is payable
	Function is locked
	Function can be accessed
!	Important functionality

```

| **BaseRegistrar** | Implementation | Ownable, IERC721 |||
| L | addController | External ! |  |NO ! |
| L | removeController | External ! |  |NO ! |
| L | setResolver | External ! |  |NO ! |
| L | nameExpires | External ! | |NO ! |
| L | available | Public ! | |NO ! |
| L | register | External ! |  |NO ! |
| L | renew | External ! |  |NO ! |
| L | reclaim | External ! |  |NO ! |
| |||||
| **BaseRegistrarImplementation** | Implementation | ERC721, BaseRegistrar |||
| L | _isApprovedOrOwner | Internal  | | |
| L | <Constructor> | Public ! |  | ERC721 |
| L | ownerOf | Public ! | |NO ! |
| L | addController | External ! |  | onlyOwner |
| L | removeController | External ! |  | onlyOwner |
| L | setResolver | External ! |  | onlyOwner |
| L | nameExpires | External ! | |NO ! |
| L | available | Public ! | |NO ! |
| L | register | External ! |  |NO ! |
| L | registerOnly | External ! |  |NO ! |
| L | _register | Internal  |  | live onlyController |
| L | renew | External ! |  | live onlyController |
| L | reclaim | External ! |  | live |
| L | supportsInterface | Public ! | |NO ! |
| |||||
| **BulkRenewal** | Implementation | |||
| L | <Constructor> | Public ! |  |NO ! |
| L | getController | Internal  | | |
| L | rentPrice | External ! | |NO ! |
| L | renewAll | External ! |  |NO ! |

```



```

| L | supportsInterface | External ! | |NO! |
|||||
| **ETHRegistrarController** | Implementation | Ownable |||
| L | <Constructor> | Public ! | ● |NO! |
| L | rentPrice | Public ! | |NO! |
| L | valid | Public ! | |NO! |
| L | available | Public ! | |NO! |
| L | makeCommitment | Public ! | |NO! |
| L | makeCommitmentWithConfig | Public ! | |NO! |
| L | commit | Public ! | ● |NO! |
| L | register | External ! | 🚫 |NO! |
| L | registerWithConfig | Public ! | 🚫 |NO! |
| L | renew | External ! | 🚫 |NO! |
| L | setPriceOracle | Public ! | ● | onlyOwner |
| L | setCommitmentAges | Public ! | ● | onlyOwner |
| L | withdraw | Public ! | ● | onlyOwner |
| L | supportsInterface | External ! | |NO! |
| L | _consumeCommitment | Internal 🔒 | ● | |
|||||
| **PriceOracle** | Interface | |||
| L | price | External ! | |NO! |
|||||
| **AggregatorInterface** | Interface | |||
| L | latestAnswer | External ! | |NO! |
|||||
| **StablePriceOracle** | Implementation | Ownable, PriceOracle |||
| L | <Constructor> | Public ! | ● |NO! |
| L | price | External ! | |NO! |
| L | setPrices | Public ! | ● | onlyOwner |
| L | premium | External ! | |NO! |
| L | _premium | Internal 🔒 | | |
| L | attoUSDToWei | Internal 🔒 | | |
| L | weiToAttoUSD | Internal 🔒 | | |
| L | supportsInterface | Public ! | |NO! |
|||||
| **NameResolver** | Implementation | |||
| L | setName | Public ! | ● |NO! |
|||||
| **ReverseRegistrar** | Implementation | Ownable, Controllable |||
| L | <Constructor> | Public ! | ● |NO! |
| L | claim | Public ! | ● |NO! |
| L | claimForAddr | Public ! | ● | authorised |
| L | claimWithResolver | Public ! | ● |NO! |
| L | claimWithResolverForAddr | Public ! | ● | authorised |
| L | setName | Public ! | ● |NO! |
| L | setNameForAddr | Public ! | ● | authorised |
| L | node | Public ! | |NO! |
| L | sha3HexAddress | Private 🔒 | | |
| L | _claimWithResolver | Internal 🔒 | ● | |
| L | ownsContract | Internal 🔒 | | |

```



```

||||| |
| **SupportsInterface** | Implementation | ISupportsInterface |||
| L | supportsInterface | Public ! | |NO! |
|||||
| **INameWrapper** | Interface | IERC1155 |||
| L | ens | External ! | |NO! |
| L | registrar | External ! | |NO! |
| L | metadataService | External ! | |NO! |
| L | names | External ! | |NO! |
| L | wrap | External ! | ● |NO! |
| L | wrapETH2LD | External ! | ● |NO! |
| L | registerAndWrapETH2LD | External ! | ● |NO! |
| L | renew | External ! | ● |NO! |
| L | unwrap | External ! | ● |NO! |
| L | unwrapETH2LD | External ! | ● |NO! |
| L | burnFuses | External ! | ● |NO! |
| L | setSubnodeRecord | External ! | ● |NO! |
| L | setSubnodeRecordAndWrap | External ! | ● |NO! |
| L | setRecord | External ! | ● |NO! |
| L | setSubnodeOwner | External ! | ● |NO! |
| L | setSubnodeOwnerAndWrap | External ! | ● |NO! |
| L | isTokenOwnerOrApproved | External ! | ● |NO! |
| L | setResolver | External ! | ● |NO! |
| L | setTTL | External ! | ● |NO! |
| L | getFuses | External ! | ● |NO! |
| L | allFusesBurned | External ! | |NO! |
|||||
| **ERC1155Fuse** | Implementation | ERC165, IERC1155, IERC1155MetadataURI |||
| L | ownerOf | Public ! | |NO! |
| L | supportsInterface | Public ! | |NO! |
| L | balanceOf | Public ! | |NO! |
| L | balanceOfBatch | Public ! | |NO! |
| L | setApprovalForAll | Public ! | ● |NO! |
| L | isApprovedForAll | Public ! | |NO! |
| L | getData | Public ! | |NO! |
| L | _setData | Internal 🔒 | ● |
| L | safeTransferFrom | Public ! | ● |NO! |
| L | safeBatchTransferFrom | Public ! | ● |NO! |
| L | _canTransfer | Internal 🔒 | ● |
| L | _mint | Internal 🔒 | ● |
| L | _burn | Internal 🔒 | ● |
| L | _doSafeTransferAcceptanceCheck | Private 🔒 | ● |
| L | _doSafeBatchTransferAcceptanceCheck | Private 🔒 | ● |
|||||
| **TestRegistrar** | Implementation | |||
| L | <Constructor> | Public ! | ● |NO! |
| L | register | Public ! | ● |NO! |
|||||
| **DefaultReverseResolver** | Implementation | |||
| L | <Constructor> | Public ! | ● |NO! |

```



```

| L | setName | Public ! | ● | onlyOwner |
|||||
| **ISupportsInterface** | Interface | |||
| L | supportsInterface | External ! | | NO ! |
|||||
| **Multicallable** | Implementation | IMulticallable, SupportsInterface |||
| L | multicall | External ! | ● | NO ! |
| L | supportsInterface | Public ! | | NO ! |
|||||
| **INameWrapper** | Interface | |||
| L | ownerOf | External ! | | NO ! |
|||||
| **PublicResolver** | Implementation | Multicallable, ABIResolver, AddrResolver,
ContentHashResolver, DNSResolver, InterfaceResolver, NameResolver, PubkeyResolver, TextResolver |||
| L | <Constructor> | Public ! | ● | NO ! |
| L | setApprovalForAll | External ! | ● | NO ! |
| L | isAuthorised | Internal 🔒 | | |
| L | isApprovedForAll | Public ! | | NO ! |
| L | supportsInterface | Public ! | | NO ! |
|||||
| **Resolver** | Interface | ISupportsInterface, IABIResolver, IAddressResolver, IAddrResolver,
IContentHashResolver, IDNSRecordResolver, IDNSZoneResolver, IInterfaceResolver, INameResolver,
IPubkeyResolver, ITextResolver |||
| L | setABI | External ! | ● | NO ! |
| L | setAddr | External ! | ● | NO ! |
| L | setAddr | External ! | ● | NO ! |
| L | setContenthash | External ! | ● | NO ! |
| L | setDnsrr | External ! | ● | NO ! |
| L | setName | External ! | ● | NO ! |
| L | setPubkey | External ! | ● | NO ! |
| L | setText | External ! | ● | NO ! |
| L | setInterface | External ! | ● | NO ! |
| L | multicall | External ! | ● | NO ! |
| L | content | External ! | | NO ! |
| L | multihash | External ! | | NO ! |
| L | setContent | External ! | ● | NO ! |
| L | setMultihash | External ! | ● | NO ! |

```



# Software Analysis

## Function Signatures

```

77372213 => setName(bytes32,string)
a7fc7a07 => addController(address)
f6a74ed7 => removeController(address)
4e543b26 => setResolver(address)
d6e4fa86 => nameExpires(uint256)
96e494e8 => available(uint256)
fca247ac => register(uint256,address,uint256)
c475abff => renew(uint256,uint256)
28ed4f6c => reclaim(uint256,address)
4cdc9549 => _isApprovedOrOwner(address,uint256)
6352211e => ownerOf(uint256)
0e297b45 => registerOnly(uint256,address,uint256)
e2be6b89 => _register(uint256,address,uint256,bool)
01ffc9a7 => supportsInterface(bytes4)
3018205f => getController()
3971d467 => rentPrice(string[],uint256)
e8d6dbb4 => renewAll(string[],uint256)
83e7f6ff => rentPrice(string,uint256)
9791c097 => valid(string)
aeb8ce9b => available(string)
f49826be => makeCommitment(string,address,bytes32)
3d86c52f => makeCommitmentWithConfig(string,address,bytes32,address,address)
f14fcbc8 => commit(bytes32)
85f6d155 => register(string,address,uint256,bytes32)
f7a16963 => registerWithConfig(string,address,uint256,bytes32,address,address)
acf1a841 => renew(string,uint256)
298f4e59 => setPriceOracle(PriceOracle)
7e324479 => setCommitmentAges(uint256,uint256)
3ccfd60b => withdraw()
50e9a715 => price(string,uint256,uint256)
50d25bcd => latestAnswer()
79cf92d3 => setPrices(uint256[])
a34e3596 => premium(string,uint256,uint256)
9732b4fc => _premium(string,uint256,uint256)
d1467f86 => attoUSDToWei(uint256)
6c01c874 => weiToAttoUSD(uint256)
1e83409a => claim(address)
4709bf4b => claimForAddr(address,address)
0f5a5466 => claimWithResolver(address,address)
0ccc4a7f => claimWithResolverForAddr(address,address,address)
c47f0027 => setName(string)
8d9522b8 => setNameForAddr(address,address,string)
bffbe61c => node(address)
27b752b8 => sha3HexAddress(address)
924192ee => _claimWithResolver(address,address,address)

```



```

85f84ea3 => ownsContract(address)
d22057a9 => register(bytes32,address)
ac9650d8 => multicall(bytes[])
a22cb465 => setApprovalForAll(address,bool)
6404a386 => isAuthorised(bytes32)
e985e9c5 => isApprovedForAll(address,address)
623195b0 => setABI(bytes32,uint256,bytes)
d5fa2b00 => setAddr(bytes32,address)
8b95dd71 => setAddr(bytes32,uint256,bytes)
304e6ade => setContenthash(bytes32,bytes)
76196c88 => setDnsrr(bytes32,bytes)
29cd62ea => setPubkey(bytes32,bytes32,bytes32)
10f13a8c => setText(bytes32,string,string)
e59d895d => setInterface(bytes32,bytes4,address)
2dff6941 => content(bytes32)
e89401a1 => multihash(bytes32)
c3d014d6 => setContent(bytes32,bytes32)
aa4cb547 => setMultihash(bytes32,bytes)
3f15457f => ens()
2b20e397 => registrar()
20c38e2b => names(bytes32)
9c50a2e9 => wrap(bytes,address,uint96,address)
a382150d => wrapETH2LD(string,address,uint96,address)
a0a5a738 => registerAndWrapETH2LD(string,address,uint256,address,uint96)
d8c9921a => unwrap(bytes32,bytes32,address)
8b4dfa75 => unwrapETH2LD(bytes32,address,address)
c1cbf66f => burnFuses(bytes32,uint96)
5ef2c7f0 => setSubnodeRecord(bytes32,bytes32,address,address,uint64)
31ea1cf9 => setSubnodeRecordAndWrap(bytes32,string,address,address,uint64,uint96)
cf408823 => setRecord(bytes32,address,address,uint64)
06ab5923 => setSubnodeOwner(bytes32,bytes32,address)
63f03326 => setSubnodeOwnerAndWrap(bytes32,string,address,uint96)
f44779b9 => isTokenOwnerOrApproved(bytes32,address)
1896f70a => setResolver(bytes32,address)
14ab9038 => setTTL(bytes32,uint64)
4ac07f41 => getFuses(bytes32)
a456f7d8 => allFusesBurned(bytes32,uint96)
00fdd58e => balanceOf(address,uint256)
4e1273f4 => balanceOfBatch(address[],uint256[])
0178fe3f => getData(uint256)
5b9f8561 => _setData(uint256,address,uint96)
f242432a => safeTransferFrom(address,address,uint256,uint256,bytes)
2eb2c2d6 => safeBatchTransferFrom(address,address,uint256[],uint256[],bytes)
a821115a => _canTransfer(uint96)
6409567a => _mint(bytes32,address,uint96)
9b1f9e74 => _burn(uint256)
084e9e24 => _doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes)
e51c223d =>
_doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes)

```



# Manual Analysis

- ❖ BaseRegistrarResolverImplementation smart contract owner can **add and remove controller**, and **set resolver**. When ownership is transferred, previous authorizations should not remain active.

```
function addController(address controller) external override onlyOwner {
    controllers[controller] = true;
function removeController(address controller) external override onlyOwner {
    controllers[controller] = false;
function setResolver(address resolver) external override onlyOwner {
    ens.setResolver(baseNode, resolver);
```

- ❖ StablePriceOracle smart contract utilizes **safemath** function to avoid common math related vulnerabilities.

InterFi

```
library SafeMath {
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a, b, "SafeMath: subtraction overflow");
    uint256 c = a * b;
    require(c / a == b, "SafeMath: multiplication overflow");
    return c;
function div(uint256 a, uint256 b) internal pure returns (uint256) {
    return div(a, b, "SafeMath: division by zero");
function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    return mod(a, b, "SafeMath: modulo by zero");
```

- ❖ Velas Domain's smart contracts do not utilize **re-entrancy guard** to prevent re-entrant calls.
- ❖ Smart contract does not call `receive()` for fallbacks. It is executed on a call to the contract with empty call data. Make sure the contract can receive token through a regular transaction, and does not throw an exception.
- ❖ DefaultReverseResolver smart contract owner can **set name** for a node.

```
function setName(bytes32 node, string memory _name) public onlyOwner(node) {
    name[node] = _name;
```





- ❖ StablePriceOracle smart contract owner can **set rent prices**.

```
function setPrices(uint[] memory _rentPrices) public onlyOwner {
    rentPrices = _rentPrices;
```

- ❖ ReverseRegistrar contract sets authorizes wallet to modify `claimForAddr()`, `claimWithResolverForAddr()`, `setNameForAddr()`.

```
modifier authorised(address addr) {
    require(
        addr == msg.sender ||
        controllers[msg.sender] ||
        ens.isApprovedForAll(addr, msg.sender) ||
        ownsContract(addr),
        "Caller is not a controller or authorised by address or the address itself"
```

- ❖ ETHRegistrarController smart contract owner can **set price oracle, commitment ages**. Arbitrary limits must be given to `setCommitmentAges()` to allow value change within the set parameters.

```
function setPriceOracle(PriceOracle _prices) public onlyOwner {
    prices = _prices;
function setCommitmentAges(uint _minCommitmentAge, uint _maxCommitmentAge) public onlyOwner {
    minCommitmentAge = _minCommitmentAge;
    maxCommitmentAge = _maxCommitmentAge;
```

- ❖ ETHRegistrarController smart contract owner can **withdraw contract balance**.

```
function withdraw() public onlyOwner {
    payable(msg.sender).transfer(address(this).balance);
```

- ❖ Smart contract has a **low severity issue** which may or may not create any functional vulnerability.

"Possible back and front-running"

- ❖ Smart contract has an **informational severity issue** which may or may not create any functional vulnerability.

"Irrelevant Code"



- ❖ Smart contract has an **informational severity issue** which may or may not create any functional vulnerability.

"Too many digits"

- ❖ Smart contract has a **low severity issue** which may or may not create any functional vulnerability.

"Floating Pragma"

- ❖ Smart contract has a **low severity issue** which may or may not create any functional vulnerability.

"Utilization of block.timestamp"

- ❖ Smart contract has a **low severity issue** which may or may not create any functional vulnerability.

"Re-entrancy"

InterFi

Smart Contract  
Security Audit



# SWC Attacks

SWC ID	Description	Status
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	! Informational
SWC-103	Floating Pragma	! Low
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELF-DESTRUCT Instruction	Passed
SWC-107	Re-entrancy	! Low
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegate Call to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	! Low
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed

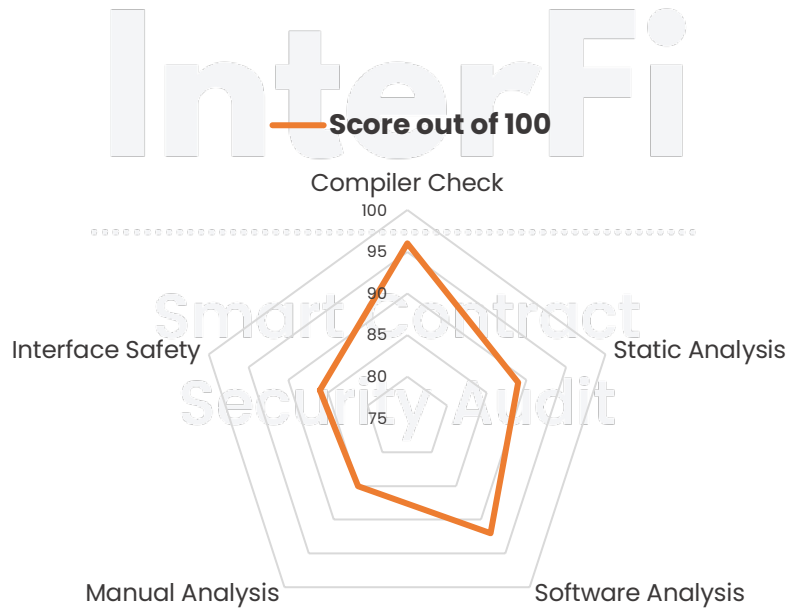


<b>SWC-119</b>	Shadowing State Variables	<b>Passed</b>
<b>SWC-120</b>	Weak Sources of Randomness from Chain Attributes	<b>Passed</b>
<b>SWC-121</b>	Missing Protection against Signature Replay Attacks	<b>Passed</b>
<b>SWC-122</b>	Lack of Proper Signature Verification	<b>Passed</b>
<b>SWC-123</b>	Requirement Violation	<b>Passed</b>
<b>SWC-124</b>	Write to Arbitrary Storage Location	<b>Passed</b>
<b>SWC-125</b>	Incorrect Inheritance Order	<b>Passed</b>
<b>SWC-126</b>	Insufficient Gas Griefing	<b>Passed</b>
<b>SWC-127</b>	Arbitrary Jump with Function Type Variable	<b>Passed</b>
<b>SWC-128</b>	DoS With Block Gas Limit	<b>Passed</b>
<b>SWC-129</b>	Typographical Error	<b>Passed</b>
<b>SWC-130</b>	Right-To-Left-Override control character (U+202E)	<b>Passed</b>
<b>SWC-131</b>	Presence of unused variables	<b>Passed</b>
<b>SWC-132</b>	Unexpected Ether balance	<b>Passed</b>
<b>SWC-133</b>	Hash Collisions With Multiple Variable Length Arguments	<b>Passed</b>
<b>SWC-134</b>	Message call with the hardcoded gas amount	<b>Passed</b>
<b>SWC-135</b>	Code With No Effects (Irrelevant/Dead Code)	<b>! Informational</b>
<b>SWC-136</b>	Unencrypted Private Data On-Chain	<b>Passed</b>



# Risk Status & Radar Chart

Risk Severity	Status
High	No high severity issues identified
Medium	No medium severity issues identified
Low	4 low severity issues identified
Informational	2 informational severity issues identified
Centralization Risk	Active contract ownership identified



## Auditor's Verdict

InterFi team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- ❖ Velas Domain's solidity source codes have **LOW RISK SEVERITY**
- ❖ Velas Domain's smart contracts have **ACTIVE OWNERSHIP**
- ❖ Velas Domain's centralization risk correlated to the active owner is **HIGH**

# InterFi

.....

### Note for stakeholders

## Smart Contract Security Audit

- ❖ Be aware that active smart contract owner privileges constitute an elevated impact on smart contract safety and security.
- ❖ If the smart contract is not deployed on any blockchain at the time of the audit, the contract can be modified or altered before blockchain development. Verify contract's deployment status in the audit report.
- ❖ Make sure that the project team's KYC/identity is verified by an independent firm.
- ❖ Always check if the contract's liquidity is locked. A longer liquidity lock plays an important role in the project's longevity. It is recommended to have multiple liquidity providers.
- ❖ Examine the unlocked token supply in the owner, developer, or team's private wallets. Understand the project's tokenomics, and make sure the tokens outside of the LP Pair are vested or locked for a longer period.



# Important Disclaimer

InterFi Network provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

## Smart Contract Security Audit

### Confidentiality

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

### No Financial Advice

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any



decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

#### Technical Disclaimer

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, INTERFI NETWORK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.





## Timeliness of content

The content contained in this audit report is subject to change without any prior notice. InterFi Network does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

## Links to other websites

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than InterFi Network. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' and social accounts' owners. You agree that InterFi Network is not responsible for the content or operation of such websites and social accounts and that InterFi Network shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

# InterFi

## Smart Contract Security Audit



# About InterFi Network

InterFi Network provides intelligent blockchain solutions. InterFi is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. **InterFi's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy to use.**

InterFi is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members, and 10+ casual contributors. **InterFi provides manual, static, and automatic smart contract analysis, to ensure that project is checked against known attacks and potential vulnerabilities.**

To learn more, visit <https://interfi.network>

To view our audit portfolio, visit <https://github.com/interfinetwork>

To book an audit, message <https://t.me/interfiaudits>





RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA 