SALUS SECURITY

MAR 2023

CODE SECURITY ASSESSMENT

ERC3525

Overview

Project Summary

- Name: ERC3525
- Version: commit 7319f4b
- Platform: EVM-compatible chains
- Language: Solidity
- Repository: https://github.com/solv-finance/erc-3525
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	ERC3525
Version	v2
Type	Solidity
Dates	Mar 10 2023
Logs	Feb 28 2023; Mar 10 2023

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	1
Total Low-Severity issues	3
Total informational issues	5
Total	11

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	ERC-721 token receiver contract may not be able to receive ERC-3525 tokens due to an issue in the _checkOnERC721Received() function	High	Business Logic	Resolved
2	ERC-3525 receiver contract may not be able to receive ERC-3525 values due to an issue in the _checkOnERC3525Received() function	High	Business Logic	Acknowledged
3	Inconsistency between the implementation and the specification for approve(uint256,address,uint256)	Medium	Business Logic	Resolved
4	Unbounded loop in _clearAprovedValues() and _existApproveValue() could lead to DoS	Low	Gas Limit	Partially Resolved
5	Issue when using solidity v0.8.0	Low	Configuration	Resolved
6	Incorrect ERC-165 identifier documented in IERC3525.sol	Low	Documentation	Resolved
7	Unnecessary payable modifier	Informational	Code Quality	Acknowledged
8	Redundant code	Informational	Redundancy	Resolved
9	Indirect import is used	Informational	Code Quality	Resolved
10	Race condition for approve(uint256,address,uint256)	Informational	Front-running	Acknowledged
11	State variable that could be declared immutable	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. ERC-721 token receiver contract may not be able to receive ERC-3525 tokens due to an issue in the _checkOnERC721Received() function

Severity: High	Category: Business Logic
Target: - contracts/ERC3525.sol	

Description

ERC-3525 tokens are designed to be compatible with ERC-721. Therefore, the user should be able to transfer an owned ERC-3525 token to a contract by using safeTransferFrom() if the contract implements the correct onERC721Received() function specified in the ERC-721 specification.

However, the _checkOnERC721Received() in the ERC3525 contract has an additional check:

contracts/ERC3525.sol:L589-L598

```
function _checkOnERC721Received(
    address from_,
    address to_,
    uint256 tokenId_,
    bytes memory data_
) private returns (bool) {
    if (to_.isContract() && IERC165(to_).supportsInterface(type(IERC721Receiver).interfaceId)) {
       try
           IERC721Receiver(to_).onERC721Received(_msgSender(), from_, tokenId_, data_)
               returns (bytes4 retval) {
           return retval == IERC721Receiver.onERC721Received.selector;
```

It requires that the receiver contract implements the IERC165 interface. If the **to_** contract lacks the **supportsInterface()** function or the fallback() function, the above highlighted line will throw.

As a result, the ERC-721 token receiver contract cannot receive tokens unless the IERC165 interface is implemented.

Proof of Concept

Contracts:

```
pragma solidity ^0.8.9;

import {ERC3525} from "@solvprotocol/erc-3525/ERC3525.sol";
import {IERC721Receiver} from
"@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import {ERC721} from "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract ERC3525TokenMock is ERC3525 {
    constructor() ERC3525("ERC3525 token", "3525", 18) {}

    function mint(address to_, uint256 slot_, uint256 amount_) external {
        _mint(to_, slot_, amount_);
    }
}

contract ERC721TokenMock is ERC721 {
    constructor() ERC721("ERC721 Token", "721") {}

    function mint(address to, uint256 tokenId) external {
        _mint(to, tokenId);
    }
}

contract ReceiverMock is IERC721Receiver {
    function onERC721Received(address, address, uint256, bytes calldata) external
returns (bytes4) {
        return IERC721Receiver.onERC721Received.selector;
    }
}
```

Test file:

```
import { expect } from "chai";
import { ethers } from "hardhat";

describe("POC", function() {
  let user, receiver, erc3525, erc721;
  const tokenId = 1;

  beforeEach(async function() {
    let deployer;
    [deployer, user] = await ethers.getSigners();

    const receiverContract = await ethers.getContractFactory(
      "ReceiverMock",
      deployer
    );
    receiver = await receiverContract.deploy();

    const erc3525Contract = await ethers.getContractFactory(
      "ERC3525TokenMock",
      deployer
    );
    erc3525 = await erc3525Contract.deploy();
    await erc3525.mint(user.address, 1, 100);
    expect(await erc3525.ownerOf(tokenId)).to.equal(user.address);

    const erc721Contract = await ethers.getContractFactory(
      "ERC721TokenMock",
      deployer
```

```
    );
    erc721 = await erc721Contract.deploy();
    await erc721.mint(user.address, tokenId);
    expect(await erc721.ownerOf(tokenId)).to.equal(user.address);
  });

  it("Should fail when transfering ERC3525 to receiver contract", async function() {
    await expect(
      erc3525
        .connect(user)
        ["safeTransferFrom(address,address,uint256)"](
          user.address,
          receiver.address,
          tokenId
        )
    ).to.be.reverted;
  });

  it("Should transfer ERC721 to receiver contract", async function() {
    await erc721
      .connect(user)
      ["safeTransferFrom(address,address,uint256)"](
        user.address,
        receiver.address,
        tokenId
      );
    expect(await erc721.ownerOf(tokenId)).to.equal(receiver.address);
  });
});
```

This PoC shows that the **ReceiverMock** contract can receive an ERC721 token that uses OpenZeppelin's ERC721 implementation, but cannot receive the ERC3525 token.

Recommendation

Consider removing the supportsInterface() check in the _checkOnERC721Received function.

Status

This issue has been resolved by the team in commit [eae9be6](eae9be6).

2. ERC-3525 receiver contract may not be able to receive ERC-3525 values due to an issue in the _checkOnERC3525Received() function

Severity: High	Category: Business Logic
Target: - contracts/ERC3525.sol	

Description

According to the ERC-3525 specification, If a smart contract wants to be informed when it receives values from other addresses, it should implement the functions in the **IERC3525Receiver** interface, which is the **onERC3525Received**() function.

contracts/ERC3525.sol:L553-L563

```
function _checkOnERC3525Received(
    uint256 fromTokenId_,
    uint256 toTokenId_,
    uint256 value_,
    bytes memory data_
) private returns (bool) {
    address to = ERC3525.ownerOf(toTokenId_);
    if (to.isContract() && IERC165(to).supportsInterface(type(IERC3525Receiver).interfaceId)) {
       try
           IERC3525Receiver(to).onERC3525Received(_msgSender(), fromTokenId_, toTokenId_,
               value_, data_) returns (bytes4 retval) {
           return retval == IERC3525Receiver.onERC3525Received.selector;
```

However, the _checkOnERC3525Received() function has an additional check for the return value of *IERC165(to).supportsInterface(type(IERC3525Receiver).interfaceId)*, if the to contract lacks the supportsInterface() function or a fallback() function, the _checkOnERC3525Received() function will revert.

As a result, the ERC-3525 token receiver contract cannot receive values unless the IERC165 interface is implemented.

Proof of Concept

Contracts:

```
pragma solidity ^0.8.9;

import {ERC3525, IERC3525Receiver} from "@solvprotocol/erc-3525/ERC3525.sol";

contract ERC3525TokenMock is ERC3525 {
    constructor() ERC3525("ERC3525 token", "3525", 18) {}

    function mint(address to_, uint256 slot_, uint256 amount_) external {
        _mint(to_, slot_, amount_);
    }
}

contract ReceiverMock is IERC3525Receiver {
    function onERC3525Received(address, uint256, uint256, uint256, bytes calldata)
external returns (bytes4) {
        return IERC3525Receiver.onERC3525Received.selector;
    }
}
```

Test file:

```
import { expect } from "chai";
import { ethers } from "hardhat";

describe("POC", function() {
  let user, receiver, erc3525;
  const tokenId1 = 1;
  const tokenId1Value = 100;
  const tokenId2 = 2;
  const tokenId2Value = 100;
  const slotMock = 1;

  beforeEach(async function() {
    let deployer;
    [deployer, user] = await ethers.getSigners();

    const receiverContract = await ethers.getContractFactory(
      "ReceiverMock",
      deployer
    );
    receiver = await receiverContract.deploy();

    const erc3525Contract = await ethers.getContractFactory(
      "ERC3525TokenMock",
      deployer
    );
    erc3525 = await erc3525Contract.deploy();
    await erc3525.mint(user.address, slotMock, tokenId1Value);
    expect(await erc3525.ownerOf(tokenId1)).to.equal(user.address);

    await erc3525.mint(receiver.address, slotMock, tokenId2Value);
    expect(await erc3525.ownerOf(tokenId2)).to.equal(receiver.address);
  });

  it("Should fail when transfering value to the receiver", async function() {
    await expect(
      erc3525
        .connect(user)
        ["transferFrom(uint256,address,uint256)"](
          tokenId1,
```

```
          receiver.address,
          tokenId1Value
        )
    ).to.be.reverted;
  });

  it("Should fail when transfering value to a token belongs to the receiver", async
function() {
    await expect(
      erc3525
        .connect(user)
        ["transferFrom(uint256,uint256,uint256)"](
          tokenId1,
          tokenId2,
          tokenId1Value
        )
    ).to.be.reverted;
  });
});
```

This PoC demonstrates that if the ERC3525 token receiver contract only implements the IERC3525Receiver interface but lacks the IERC165 interface, one cannot transfer values to it by using transfer(uint256,address,uint256), and one cannot transfer values to the token owned by it by using transfer(uint256,uint256,uint256).

Recommendation

Consider removing the supportsInterface() check in the _checkOnERC3525Received() function.

Status

This issue has been acknowledged by the team. Considering the usage scenarios and the clarity of implementation, the team has decided to use ERC-165's supportsInterface() function to check the existence of onERC3525Received() in the receiver contract.

3. Inconsistency between the implementation and the specification for approve(uint256,address,uint256)

Severity: Medium	Category: Business Logic

Target:
- contracts/ERC3525.sol

Description

contracts/IERC3525.sol:L63-L76

```
/**
 * @notice Allow an operator to manage the value of a token, up to the `_value` amount.
 * @dev MUST revert unless caller is the current owner, an authorized operator, or the approved
 *  address for `_tokenId`.
 *  MUST emit ApprovalValue event.
 * @param _tokenId The token to approve
 * @param _operator The operator to be approved
 * @param _value The maximum value of `_toTokenId` that `_operator` is allowed to manage
 */
function approve(
    uint256 _tokenId,
    address _operator,
    uint256 _value
) external payable;
```

The NatSpec documentation for the approve(uint256,address,uint256) function in the IERC3525 contract indicates that the approved address for _tokenId should be able to use approve(uint256,address,uint256) to manage the value of _tokenId. In other words, a token ID level authority should be able to use a value level approve function.

contracts/ERC3525.sol:L151-L161

```
function approve(uint256 tokenId_, address to_, uint256 value_) public payable virtual override {
    address owner = ERC3525.ownerOf(tokenId_);
    require(to_ != owner, "ERC3525: approval to current owner");

    require(
        _msgSender() == owner || ERC3525.isApprovedForAll(owner, _msgSender()),
        "ERC3525: approve caller is not owner nor approved for all"
    );

    _approveValue(tokenId_, to_, value_);
}
```

However, a token ID level authority cannot bypass the check in the approve(uint256,address,uint256) function. Only the owner or the full level authority can use this value level approve() function.

Recommendation

Consider using *ERC3525._isApprovedOrOwner(_msgSender(), tokenId_)* to replace the *_msgSender() == owner || ERC3525.isApprovedForAll(owner, _msgSender())* check.

Status

This issue has been resolved by the team in commit af107ee.

4. Unbounded loop in _clearAprovedValues() and _existApproveValue() could lead to DoS

Severity: Low	Category: Gas Limit
Target: - contracts/ERC3525.sol	

Description

contracts/ERC3525.sol:L454-L461

```
function _clearApprovedValues(uint256 tokenId_) internal virtual {
    TokenData storage tokenData = _allTokens[_allTokensIndex[tokenId_]];
    uint256 length = tokenData.valueApprovals.length;
    for (uint256 i = 0; i < length; i++) {
        address approval = tokenData.valueApprovals[i];
        delete _approvedValues[tokenId_][approval];
    }
}
```

When transferring tokenId using

- transferFrom(address,address,uint256),
- safeTransferFrom(address,address,uint256),
- safeTransferFrom(address,address,uint256,bytes),

or burning tokenId using

- _burn(),

all these functions need to call the _clearApprovedValues() function to clear the approved values in the _approvedValues state variable. In the above loop, the gas cost will increase as the length of the valueApprovals array increases. This may cause a potential DoS for transferFrom(address,address,uint256), safeTransferFrom()s, and _burn(); as more and more addresses are approved for a token, the gas cost of _clearApprovedValues() exceeds the block gas limit, then the related functions become unavailable.

contracts/ERC3525:sol:L463-L471

```
function _existApproveValue(address to_, uint256 tokenId_) internal view virtual returns
(bool) {
    uint256 length = _allTokens[_allTokensIndex[tokenId_]].valueApprovals.length;
    for (uint256 i = 0; i < length; i++) {
        if (_allTokens[_allTokensIndex[tokenId_]].valueApprovals[i] == to_) {
            return true;
        }
    }
    return false;
}
```

The same issue also applies to the _existApproveValue() function. The _existApproveValue() is called by approve(uint256,address,uint256) and transferFrom(uint256,address,uint256). Therefore they might be unavailable because the unbounded loop could run out of gas.

Even worse, the **valueApprovals** array is not cleaned when a token is transferred from one address to another, making the DoS more possible.

Recommendation

For the _existApproveValue() function, consider using OpenZeppelin's EnumerableSet instead of uint256[] for valueApprovals to avoid the unbounded loop.

For the _clearApprovedValues() function, consider also clearing the valueApprovals to avoid the length from becoming too large.

Status

The issue with the _clearApprovedValues() function has been resolved by the team by clearing the valueApprovals in commit eae9be6.

The issue with the _existApproveValue() function has been acknowledged by the team. The team has decided to not use EnumerableSet since it incurs higher storage cost and may not be a better solution.

5. Issue when using solidity v0.8.0

Severity: Low	Category: Configuration
Target: - contracts/ERC3525.sol	

Description

Since OpenZeppelin v4.5.0, the Solidity pragma for Address.sol is increased from ^0.8.0 to ^0.8.1.

package.json:L61

```
"@openzeppelin/contracts": "^4.7.3",
```

The version of the OpenZepplin library used in the ERC3525 project is above v4.5.0

contracts/ERC3525.sol:L8

```
import "@openzeppelin/contracts/utils/Address.sol";
```

contracts/ERC3525.sol:L3

```
pragma solidity ^0.8.0;
```

However, the ERC3525 contract uses the Address library, but specifies the Solidity pragma ^0.8.0. When a developer builds on the ERC3525 contract and uses Solidity v0.8.0, the contract will fail to compile.

Recommendation

Consider increasing the Solidity pragma from ^0.8.0 to ^0.8.1 in the contracts that depend on Address.sol.

Status

This issue has been resolved by the team by increasing the Solidity pragma from ^0.8.0 to ^0.8.1 in commit eae9be6.

6. Incorrect ERC-165 identifier documented in IERC3525.sol

Severity: Low	Category: Documentation
Target: - contracts/ERC3525.sol	

Description

The ERC-165 identifier for the IERC3525 interface should be **0xd5358140**, which is the value of *type(IERC3525).interfaceId*. This value is also documented in the ERC-3525 specification.

contracts/IERC3525.sol:L11

```
* Note: the ERC-165 identifier for this interface is 0xc97ae3d5.
*/
interface IERC3525 is IERC165, IERC721 {
```

However, the note in the IERC3525.sol file says the ERC165 identifier is 0xc97ae3d5, which is not correct.

Recommendation

Consider fixing the ERC-165 identifier documented in the IERC3525.sol file.

Status

This issue has been resolved by the team in commit eae9be6.

2.3 Informational Findings

7. Unnecessary payable modifier

Severity: Informational	Category: Code Quality

Target:
- contracts/ERC3525.sol

Description

ERC3525.sol:L151

```
function approve(uint256 tokenId_, address to_, uint256 value_) public payable
```

ERC3525.sol:L223

```
function approve(address to_, uint256 tokenId_) public payable
```

ERC3525.sol:L172

```
function transferFrom(
    uint256 fromTokenId_,
    address to_,
    uint256 value_
) public payable
```

ERC3525.sol:L186

```
function transferFrom(
    uint256 fromTokenId_,
    uint256 toTokenId_,
    uint256 value_
) public payable
```

ERC3525.sol:L200

```
function transferFrom(
    address from_,
    address to_,
    uint256 tokenId_
) public payable
```

ERC3525.sol:L210

```
function safeTransferFrom(
    address from_,
    address to_,
    uint256 tokenId_,
    bytes memory data_
) public payable
```

ERC3525.sol:L219

```
function safeTransferFrom(
    address from_,
    address to_,
    uint256 tokenId_
) public payable
```

Since there is no withdrawal mechanism in the ERC3525 contract, none of the above functions should be marked as payable. If one accidentally sends ETHs through these payable functions, the funds are locked in the contract.

Recommendation

Consider removing the payable modifier, similar to what OpenZepplin did with its ERC-721 implementation.

Status

This issue has been acknowledged by the team. The team has decided to leave it as the payable modifier may be required by the application scenarios.

8. Redundant code

Severity: Informational	Category: Redundancy

Target:
- contracts/ERC3525.sol

Description

ERC3525.sol:L102-L106

```
function ownerOf(uint256 tokenId_) public view virtual override returns (address owner_)
{
    _requireMinted(tokenId_);
    owner_ = _allTokens[_allTokensIndex[tokenId_]].owner;
    require(owner_ != address(0), "ERC3525: invalid token ID");
}
```

The ownerOf() function has a **_requireMinted(tokenId_)** line that checks whether the tokenId_ is minted.

ERC3525.sol:L274-L282

```
function _isApprovedOrOwner(address operator_, uint256 tokenId_) internal view virtual
returns (bool) {
    _requireMinted(tokenId_);
    address owner = ERC3525.ownerOf(tokenId_);
    return (
        operator_ == owner ||
        ERC3525.isApprovedForAll(owner, operator_) ||
        ERC3525.getApproved(tokenId_) == operator_
    );
}
```

ERC3525.sol:L317-L325

```
function _mintValue(uint256 tokenId_, uint256 value_) internal virtual {
    _requireMinted(tokenId_);

    address owner = ERC3525.ownerOf(tokenId_);
    uint256 slot = ERC3525.slotOf(tokenId_);
    _beforeValueTransfer(address(0), owner, 0, tokenId_, slot, value_);
    __mintValue(tokenId_, value_);
    _afterValueTransfer(address(0), owner, 0, tokenId_, slot, value_);
}
```

Thus, the _requireMinted(tokenId_) line in _isApprovedOrOwner() and _mintValue() is redundant because it is also checked in the following ERC3525.ownerOf() function.

Recommendation

Consider removing the redundant **_requireMinted(tokenId_)** line.

Status

This issue has been resolved by the team in commit eae9be6.

9. Indirect import is used

Severity: Informational	Category: Code Quality
Target: - contracts/ERC3525.sol	

Description

ERC3525.sol:L68

```
interfaceId == type(IERC165).interfaceId ||
```

ERC3525.sol:L6

```
import "@openzeppelin/contracts/utils/introspection/ERC165.sol";
```

The IERC165 interface is used in the ERC3525 contract, but ERC165.sol is imported. The IERC165 interface is indirectly imported from the ERC165.sol file.

Recommendation

Consider changing `"@openzeppelin/contracts/utils/introspection/ERC165.sol"` to `"@openzeppelin/contracts/utils/introspection/IERC165.sol"`

Status

This issue has been resolved by the team in commit eae9be6.

10. Race condition for approve(uint256,address,uint256)

Severity: Informational	Category: Front-running
Target: - contracts/ERC3525.sol	

Description

Since there is no direct way to increase and decrease allowance relative to its current value, the function approve(uint256,address,uint256) has a race condition similar to one of ERC-20 approvals. Further details regarding the race condition can be found here.
Simply put, the **approve()** function creates the potential for an approved spender to spend more than the intended amount. A front running attack can be used to enable an approved spender to call **transferFrom()** both before and after the call to **approve()** is processed.

Recommendation

Consider adding increaseAllowance() and decreaseAllowance() functions for ERC-3525 tokens, similar to what OpenZeppelin did with its ERC-20 implementation.

Status

This issue has been acknowledged by the team.

11. State variable that could be declared immutable

Severity: Informational	Category: Gas Optimization
Target: - contracts/ERC3525.sol	

Description

ERC3525.sol:L44

```
uint8 private _decimals;
```

Since the state variable **_decimals** will have a fixed value after the contract is deployed, it could be declared immutable to save gas.

Recommendation

Consider adding the immutable modifier to the _decimals state variable.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in commit 7319f4b:

File	SHA-1 hash
contracts/ERC3525SlotApprovable.sol	88ed30ceac27b9ef885615709f3b7589b453ba95
contracts/ERC3525Mintable.sol	3185ef381e9d31b7804a7582851fe5778eff4a75
contracts/ERC3525.sol	d06d7e2d7653e275fafe6355d90f7d26c8aeb1e9
contracts/ERC3525Burnable.sol	0602dd9b07f41feb17084f15026a67e9ad6e23b2
contracts/ERC3525SlotEnumerable.sol	ae9eb8bfe435ea393fb5bffa96d6c7fe4650e0d2