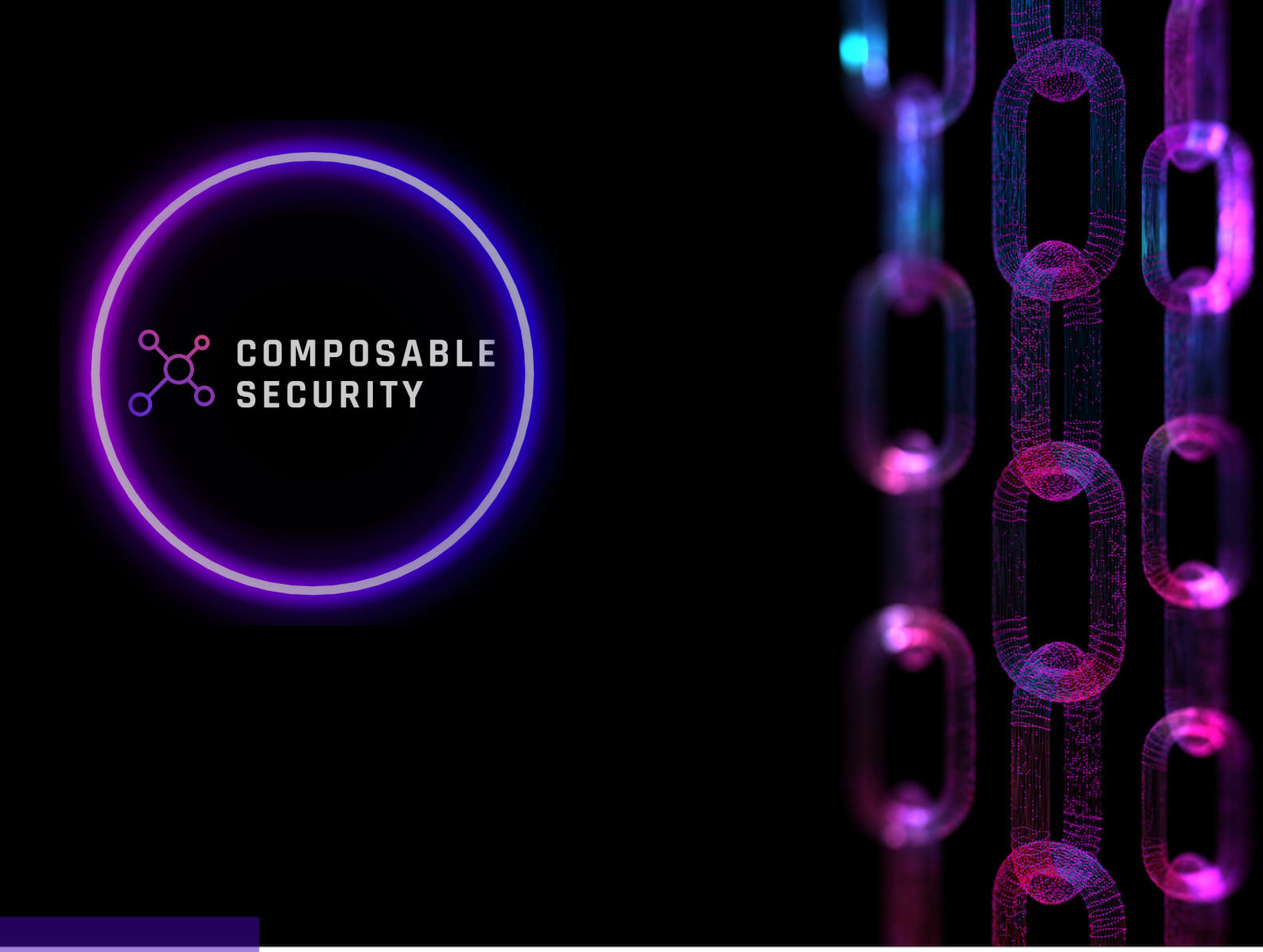


COMPOSABLE SECURITY

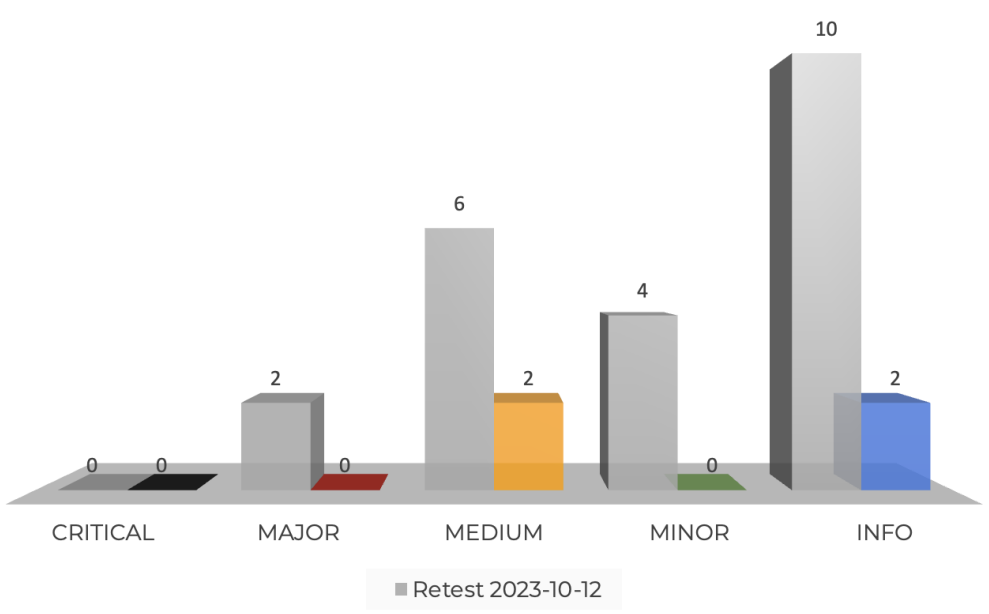
REPORT

Smart contract security review for Research Portfolio

Prepared by: Composable Security
Report ID: RSPF-ed40cad
Test time period: 2023-08-15 - 2023-08-18
Retest time period: 2023-09-14 - 2023-10-12
Report date: 2023-10-10
Version: 1.1
Visit: composable-security.com

Contents

1. Retest summary (2023-10-12)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team was involved in a one-time iteration of verification whether the vulnerabilities detected during the tests (between 2023-08-15 and 2023-08-18) were removed correctly and no longer appear in the code.

The current status of detected issues is as follows:

- 2 **major** vulnerabilities were fixed.
- 6 vulnerabilities with a **medium** impact on risk were handled as follows:
  - 4 has been fixed,
  - 1 has been partially fixed,
  - 1 has been acknowledged.
- all 4 vulnerabilities with a **minor** impact on risk were fixed.
- 10 security **recommendations** were handled as follows:
  - 8 have been implemented,
  - 1 has been partially implemented,
  - 1 has been acknowledged.
- Conversations during the retest allowed Composable Security team to classify a previously insignificant deviation from best practices for system security as a vulnerability with a medium impact on risk (RSPF-ed40cad-M06). The team discovered that in addition to the support for `ResearchToken` and `USDC` indicated in the documentation, the

distribution of other tokens (including `USDT`, which would generate problems) is also acceptable (even though, it is not considered currently by the team). However, we recommend fixing it if other stablecoins are considered in the future.

- The developers team decided to add `withdraw` and `withdrawAmount` functions in the distributor contract (commits: `a5e7722`, `4147de2`; out of scope of the retests) to withdraw undistributed tokens in an emergency case. The function can be called only by the Research Portfolio GnosisSafe address and returned to a specific address, both defined during initialization and immutable.

1.2. Scope

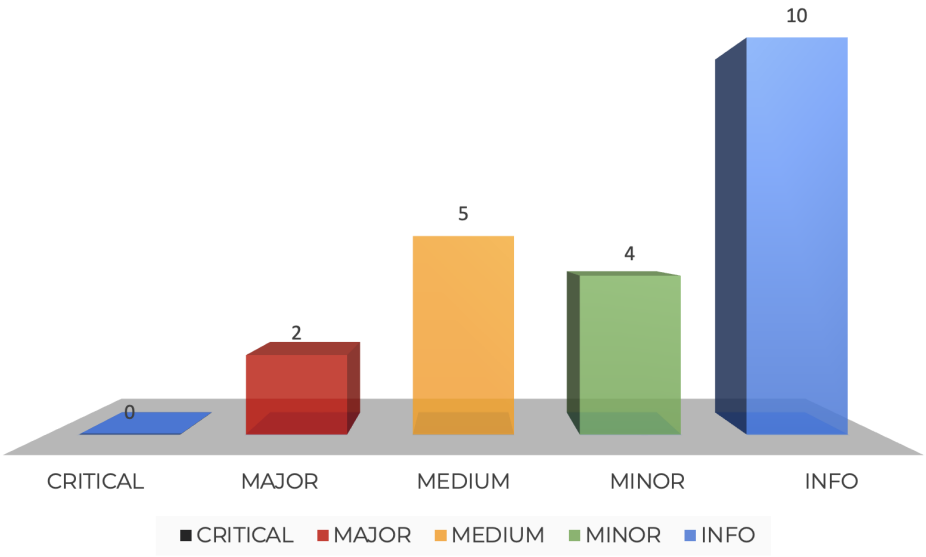
The retest scope included the same contracts, on a different commit in the same repository.

GitHub repository: https://github.com/researchportfolio/researchportfolio
CommitID: 591a2a1e6605da1cb0642d1b58e108dd375b6db4

2. Current findings status

ID	Severity	Vulnerability	Status
RSPF-ed40cad-H01	MAJOR	Skipping Entrypoint to bypass business logic	FIXED
RSPF-ed40cad-H02	MAJOR	Blocking creation of valid ResearchToken	FIXED
RSPF-ed40cad-M01	MEDIUM	Use of *tx.origin* from access control	FIXED
RSPF-ed40cad-M02	MEDIUM	Anyone can mint research tokens	FIXED
RSPF-ed40cad-M03	MEDIUM	Anyone can set distributor's token	FIXED
RSPF-ed40cad-M04	MEDIUM	Lack of distributor initial parameters validation	FIXED
RSPF-ed40cad-M05	MEDIUM	Inconsistent event emitting	PARTIALLY FIXED
RSPF-ed40cad-M06	MEDIUM	Lack of *safeTransfer* for transfer of distributed tokens	ACKNOWLEDGED
RSPF-ed40cad-L01	MINOR	The number of tokens may exceed 1 million	FIXED
RSPF-ed40cad-L02	MINOR	The distribution amounts are not immediately verified	FIXED
RSPF-ed40cad-L03	MINOR	Lack of check for the same sizes	FIXED
RSPF-ed40cad-L04	MINOR	Invalid parameter name	FIXED
ID	**Severity**	**Recommendation**	**Status**
RSPF-ed40cad-R01	INFO	Consider using the latest solidity version	NOT IMPLEMENTED
RSPF-ed40cad-R02	INFO	Consider changing variable name	IMPLEMENTED
RSPF-ed40cad-R03	INFO	Consider changing *freeze* function name	IMPLEMENTED
RSPF-ed40cad-R04	INFO	Consider removing Context inheritance	IMPLEMENTED
RSPF-ed40cad-R05	INFO	Remove left-over code from ERC1155	IMPLEMENTED
RSPF-ed40cad-R06	INFO	Consider using existing libraries	NOT IMPLEMENTED
RSPF-ed40cad-R07	INFO	Gas optimization: Inefficient state variable increment	IMPLEMENTED
RSPF-ed40cad-R08	INFO	Gas optimization: Use custom errors instead of require statements.	IMPLEMENTED
RSPF-ed40cad-R09	INFO	Gas optimization: Cache the array length outside a loop	IMPLEMENTED
RSPF-ed40cad-R10	INFO	Gas optimization: Redundant validation	IMPLEMENTED

3. Security review summary (2023-08-18)



3.1. Results

The **Research Portfolio** engaged Composable Security to review security of **researchportfolio application**. The team was engaged and the communication was very good. Composable Security conducted this assessment over 1 person-week with 2 engineers.

The summary of findings is as follows:

- **2** vulnerabilities with a **major** impact on risk were identified. Their potential consequence are:
  - Theft of donations from philanthropists (by fooling them).
  - Inability to deploy the research token by its authenticated author (minter).
- **5** vulnerabilities with a **medium** impact on risk were identified.
- **4** vulnerabilities with a **minor** impact on risk were identified.
- **10 recommendations** have been proposed that can improve overall security and help implement best practice.
- The most important issues detected concern the possibility of disrupting business flow through direct communication with the factory smart contracts.
- During the security review, a major vulnerability was detected in the web application supporting the verification process of research papers. The vulnerability note was posted in a separate file as it was not part of the audit.

Composable Security recommends that **Research Portfolio** complete the following:

- Address all reported issues.
- Consider conducting an full audit of the web application part.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.

- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Review dependencies and upgrade to the latest stable versions.

3.2. Scope

The scope of the tests included selected contracts from the following repository.

GitHub repository: https://github.com/researchportfolio/researchportfolio
CommitID: ed40cada20b7e07519be4606e8b33dccf05124ae

The detailed scope of tests can be found in Agreed scope of tests.

4. Project details

4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **Research Portfolio** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

The subjects of the test were selected contracts from the **Research Portfolio** repository.

GitHub repository:
https://github.com/researchportfolio/researchportfolio
CommitID: ed40cada20b7e07519be4606e8b33dccf05124ae

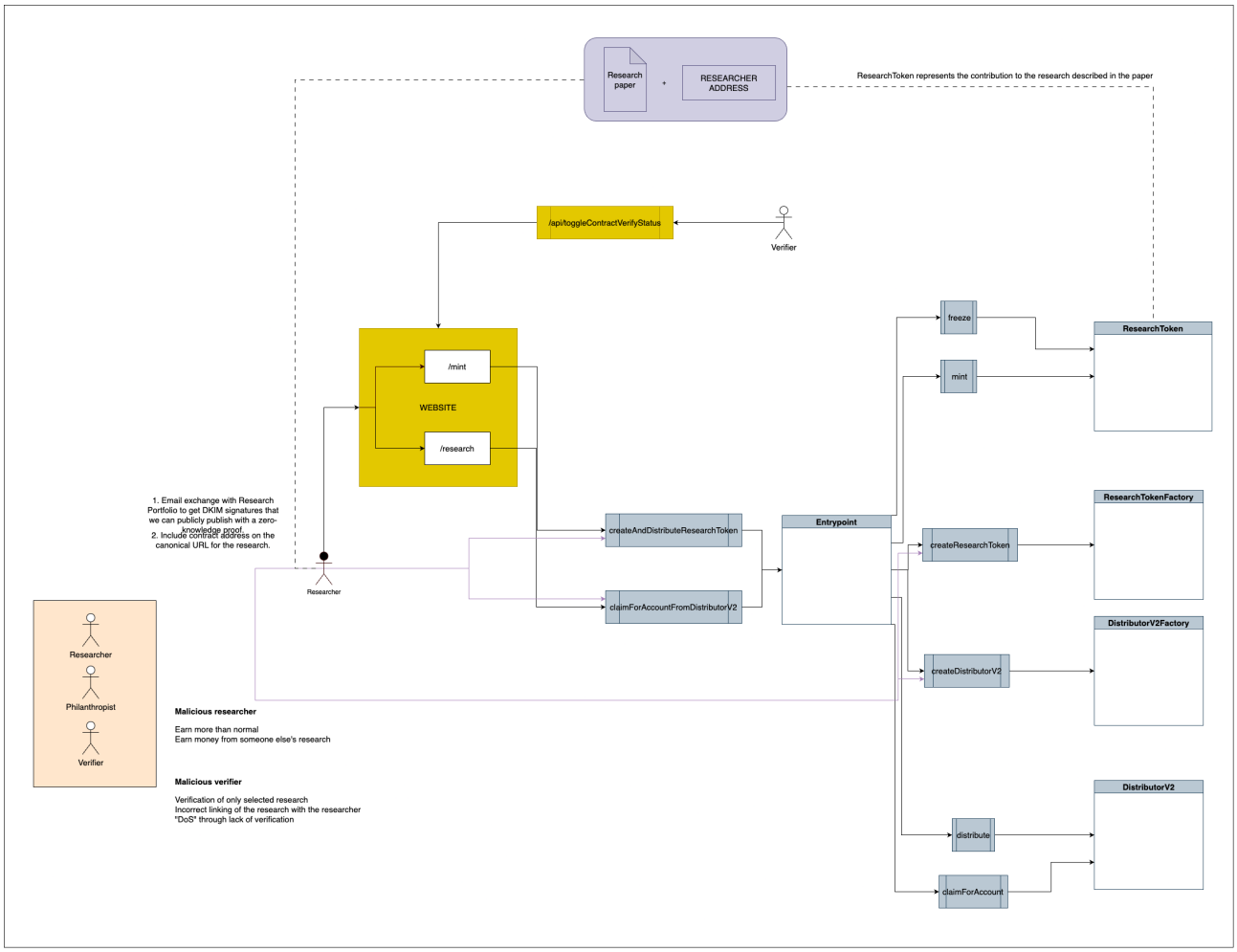
Files in scope:

```
├── ./abstracts
│   └── ./abstracts/ERC20Abstract.sol
├── ./distributor
│   ├── ./distributor/DistributorV2.sol
│   └── ./distributor/DistributorV2Factory.sol
├── ./entry
│   └── ./entry/Entrypoint.sol
├── ./initializable
│   └── ./initializable/Initializable.sol
├── ./interfaces
│   ├── ./interfaces/IDistributorV2.sol
│   ├── ./interfaces/IERC20Distributor.sol
│   └── ./interfaces/IResearch.sol
├── ./libraries
│   └── ./libraries/Clones.sol
└── ./token
    ├── ./token/MerkleDistribution.sol
    ├── ./token/ResearchToken.sol
    ├── ./token/ResearchTokenFactory.sol
    └── ./token/ResearchTokenInput.sol
```

Documentation: The business flows were explained by the team and described with simple diagrams. Additional source of information: `https://www.researchportfolio.co/faq`

4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.



Potential attackers goals:

- Theft of philanthropist funds.
- Lock researcher funds in the contract.
- Theft of other researcher funds.
- Unfair distribution of funds.
- Block the contract, so that others cannot use it.
- Impersonating other researchers.

Potential scenarios to achieve the indicated attacker's goals:

- Impersonating research paper author.
- Verification of only selected research papers.
- Incorrect linking of the research paper with the researcher.
- Influence or bypass the business logic of the system.
- Take advantage of arithmetic errors.
- Minting more research tokens than promised for distribution.
- Privilege escalation through incorrect access control to functions or badly written modifiers.
- Existence of known vulnerabilities (e.g., front-running, re-entrancy).
- Design issues.
- Excessive power, too much in relation to the declared one.
- Poor security against taking over the managing account.
- Private key compromise, rug-pull.
- Withdrawal of more funds than expected.
- Modifying or executing submitted transactions.

4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **Research Portfolio** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests using slither.
- Custom scripts (e.g. unit tests) to verify scenarios from initial threat modeling.
- **Manual review of the code.**

4.5. Disclaimer

Smart contract security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Vulnerabilities

[RSPF-ed40cad-H01] Skipping Entrypoint to bypass business logic

MAJOR FIXED

Retest (2023-10-12)

The team decided to keep the `Entrypoint` contract, but access to the `createResearchToken` function from `ResearchTokenFactory` and to `createDistributorV2` from `DistributorV2Factory` contract has been restricted only for the `Entrypoint` contract as recommended.

Affected contracts

- ResearchTokenFactory.sol#L56-72
- DistributorV2Factory.sol#L22-33

Description

The `ResearchTokenFactory` and `DistributorV2Factory` contracts are meant to be called indirectly through the `Entrypoint` contract, that creates a `ResearchToken`, and then mints it to recipients and distributes it to claimers.

```
26 // This function creates a ResearchToken, then mints it to recipients and
27 // distributes it to claimers.
28 // Args:
29 //  input: A ResearchTokenInput struct.
30 //  merkleDistribution: A MerkleDistribution struct.
31 //  addresses: a list of addresses to distribute to.
32 //  amounts: a list of amounts to distribute; same length as addresses.
33 // Returns:
34 //  the address of the research token.
35 //  the address of the distributor, possibly 0x0 if nothing to distribute.
36 function createAndDistributeResearchToken(
37     ResearchTokenInput memory input,
38     MerkleDistribution memory merkleDistribution,
39     address[] memory addresses,
40     uint256[] memory amounts,
41     bool doDistributorEventEmit
```

```
42 ) external returns (address, address) {
43     address minter = msg.sender;
44
45     address token = tokenFactory.createResearchToken(minter, input, false);
46
47     address distributor;
48     if (merkleDistribution.amount > 0) {
49         distributor = distributorFactory.createDistributorV2(
50             merkleDistribution.merkleRoot,
51             merkleDistribution.manifest,
52             false
53         );
54         if (doDistributorEventEmit) {
55             emit CreateDistributorV2(distributor);
56         }
57         ResearchToken(token).mint(distributor, merkleDistribution.amount);
58         DistributorV2(distributor).distribute(address(token));
59     }
60
61     for (uint i = 0; i < addresses.length; i++) {
62         ResearchToken(token).mint(addresses[i], amounts[i]);
63     }
64
65     ResearchToken(token).freeze();
66
67     emit CreateResearchToken(token);
68
69     return (token, distributor);
70 }
```

As we can see, *Entrypoint* not only calls factories with the parameters passed but also imposes some restrictions and introduces verifications. These restrictions do not apply when the attacker calls the factories directly.

It allows for much greater freedom in business flow and bypasses existing checks. Especially due to the fact that tokens are not automatically minted as in the case of invocation from the `Entrypoint` level.

Attack scenario

The attackers might take the following steps in turn:

① Call `createResearchToken` in `ResearchTokenFactory`, giving their address as minter.
② Call `createDistributorV2` in `DistributorV2Factory` setting any distribution (e.g. 50% for

them and 25% for two researchers).

③ The site catches their `ResearchToken` as such with the distribution.

④ The attacker passes the verification, because the research might be actually theirs (*Due to the detected web vulnerability, the attacker can verify any research paper, not necessarily their own.*)

⑤ People willingly donate to their research because they assume that research author would share prize with 2 other researchers.

⑥ No token has been minted so far because they haven't used `EntryPoint`. They mint all the tokens for themselves and get the whole prize.

Result of the attack: Theft of donations from philanthropists (by fooling them).

Recommendation

- Call the `Entrypoint` directly from the front-end and restrict access to the `ResearchTokenFactory` and `DistributorV2Factory` for `Entrypoint` contract only by appropriate modifiers.
  - ResearchTokenFactory.sol#L56-72
  - DistributorV2Factory.sol#L22-33
- Consider moving the mint and distribution parts to factory contracts so that `Entrypoint` would only be a facilitation to pass parameters and call them in the right order.
  Alternatively, as the `Entrypoint` contract unnecessarily increases gas consumption, consider using the factories directly from the front-end application.

References

1. SCSVS G4: Business logic

[RSPF-ed40cad-H02] Blocking creation of valid ResearchToken

MAJOR FIXED

Retest (2023-10-12)

The vulnerability has been removed as recommended.

Affected contracts

- ResearchTokenFactory.sol#L56
- Entrypoint.sol#L45

Description

The `createResearchToken` accepts two parameters (namely `researchIdentifier` and `minter`) that are used to generate salt and determine the address of the research token contract.

Additionally, as mentioned in the vulnerability RSPF-ed40cad-H01, the attacker is able to bypass the `Entrypoint` contract and call the `createResearchToken` function directly.

Those to facts lead to a situation where the attacker is able to front-run the researcher, deploy a contract on his behalf and block the researcher from deploying his contract.

Note: The severity of this vulnerability has been set to MAJOR because one of the mechanism that authenticates the address of the author (and deployer) is to put the address in the research paper's metadata. That is hard to quickly change and therefore the researcher is not able to easily use different `minter` address.

Attack scenario

The attackers might take the following steps in turn:

① The researcher is calling the createAndDistributeResearchToken function to create a research token and distributor.
② The attacker reads the `researchIdentifier` and `minter` parameters from the transaction and front-runs it with a direct call to createResearchToken function with those parameters.
③ The researcher's transaction reverts and he cannot deploy the research token with the same identifier and minter address.

Result of the attack: Inability to deploy the research token by its authenticated author (minter).

Recommendation

Follow the recommendations from RSPF-ed40cad-H01 and make sure that the createResearchToken function can be called only by the `Entrypoint` contract.

References

1. G5: Access control

[RSPF-ed40cad-M01] Use of *tx.origin* from access control

MEDIUM FIXED

Retest (2023-10-12)

The vulnerability has been removed as recommended. The function's name has been changed to `freezeMinting` (as recommended in RSPF-ed40cad-R03) and the requirement has been removed. Now, the access to the function is not protected with any modifier, but it is always called right after the deployment and minting of the token, as recommended.

Affected contracts

- ResearchToken.sol#L41

Description

In the `ResearchToken` smart contract, the `freeze` function is using `tx.origin` to validate if the caller has minter privileges.

```
39 function freeze() public {
40     require(
41         tx.origin == minter,
42         "ResearchToken: Can only be frozen by minter"
43     ); // XXX tx.origin a bad choice.
44     require(!frozen, "ResearchToken: Already frozen");
45     require(
46         totalSupply() == erc20_cap,
47         "ResearchToken[ERC20]: WRONG_MINTED_TOKEN_COUNT"
48     );
49     frozen = true;
50 }
```

It blocks legitimate usage if a contract address (e.g. multi-sig) is specified as the minter. The transaction will fail at the end of `createAndDistributeResearchToken` function during `freeze` execution and the user will pay for the gas.

In the case of `tx.origin`, it is also possible to deceive users and perform operations on their behalf via a malicious contract, but in this particular case the risk associated with it is lower than the problems resulting from the limited functioning.

Vulnerable scenario

The vulnerable scenario might look like this:

① The researcher uses multi-sig contract as the minter and creates a research token contract for their research paper (via `Entrypoint` contract).
② At the very end the `freeze` function fails because of `tx.origin` validation, so the whole

transaction reverts, but the researcher pays for most of the operations.

Result of the attack: Blocking the possibility of using contracts (e.g. multi-sig) as minter by researchers.

Additionally, due to the nature of `tx.origin` the vulnerability allows to call the `freeze` function on behalf of the minter. This potential result makes sense with the additional exploit of vulnerability RSPF-ed40cad-H01, which allows not to freeze the token immediately.

Recommendation

- Do not use `tx.origin`. Allow to call the `freeze` function only by the `Entrypoint` contract and make sure it is called **always** right after the research token is deployed and minted.
  - ResearchToken.sol#L41

References

1. G5: Access control
2. G8: Denial of service

[RSPF-ed40cad-M02] Anyone can mint research tokens

MEDIUM FIXED

Retest (2023-10-12)

The vulnerability has been removed by implementing the changes recommended in RSPF-ed40cad-H01.

Affected contracts

- ResearchToken.sol#L52-62

Description

The `mint` function is not restricted to any role, anyone can call it after creating a `ResearchToken` .

Current security mechanism relays on the fact that tokens should be completely minted at the time of `ResearchToken` creation, but it is possible to refer directly to `ResearchTokenFactory` and then this security assumption is bypassed.

```
52 function mint(address account, uint256 amount) public {
53     require(
```

```
54          !frozen,
55          "ResearchToken[ERC20]: Minting is not allowed once frozen"
56      );
57      require(
58          totalSupply() + amount <= erc20_cap,
59          "ResearchToken[ERC20]: erc20_cap exceeded"
60      );
61      _mint(account, amount);
62  }
```

Attack scenario

The attackers might take the following steps in turn:

① The researcher creates their `ResearchToken` via `ResearchTokenFactory`.
② The attacker calls the `mint` function on that token to mint more tokens to attacker's address.

Result of the attack: Unlimited and unauthorized mint of research tokens.

Recommendation

Restrict access to the `mint` function only to the `Entrypoint` contract.

References

1. SCSVS G5: Access control

[RSPF-ed40cad-M03] Anyone can set distributor's token

MEDIUM FIXED

Retest (2023-10-12)

The function's name has been changed to `setToken`. It is not protected directly, but whenever the `Entrypoint` creates a distributor, it always calls the `setToken` function in the same call, as recommended.

Affected contracts

- DistributorV2.sol#L54

Description

The `distribute` function is not authorized and anyone can call this function to assign any token to the `DistributorV2` contract.

```
54 function distribute(address token_) external override {
55     require(token == address(0x0), "Token has already been set.");
56     require(
57         token_ != address(0x0),
58         "Target token should not be zero address."
59     );
60     token = token_;
61     emit ERC20Distribution(token_, DistributorTypeMerkleV2);
62 }
```

The `distribute` function is called in the `createAndDistributeResearchToken` function (from `Entrypoint` contract) right after the deployment of the distribution.

However, due to the vulnerability RSPF-ed40cad-H01, the researcher is able to create research token and distributor directly by calling functions in the corresponding factory contracts. If the researcher forgets to set the token in distributor contract, anyone can do it and lock the research tokens transferred to the distributor.

Attack scenario

① The researcher mints a distributor contract.
② The researcher mints the research token contract and transfers some tokens to the distributor.
③ The researcher forgets to call the `distribute` function or is front-run.
④ The attacker calls the `distribute` function and sets their fake token.

Result of the attack: The research tokens transferred to the distributor are stuck.

Recommendation

- Follow the recommendations from RSPF-ed40cad-H01 and make sure that the `distribute` function is called **always** after the distributor contract is deployed.
- Consider changing the function name from `distribute` to `setToken`.

References

1. G5: Access control

[RSPF-ed40cad-M04] Lack of distributor initial parameters validation

MEDIUM FIXED

Retest (2023-10-12)

The vulnerability has been removed as recommended.

Affected contracts

- DistributorV2.sol#L42

Description

The parameters of distrubutor's function `__DistributorV2_init_unchained` are not validated on-chain.

```
42 function __DistributorV2_init_unchained(
43     bytes32 merkleRoot_,
44     string memory manifest_
45 ) internal initializer {
46     merkleRoot = merkleRoot_;
47     _manifest = manifest_;
48 }
```

The distributor contract can be deployed with empty `merkleRoot` and `manifest` parameters and the transferred tokens would be stuck on it.

Result of the attack: The research tokens transferred to the distributor are stuck.

Recommendation

Validate the values of `merkleRoot` (non-zero) and `manifest` (non-empty) at the beginning of the function and revert if they are invalid.

References

1. SCSVS G5: Access control

[RSPF-ed40cad-M05] Inconsistent event emitting

MEDIUM PARTIALLY FIXED

Retest (2023-10-12)

The issues mentioned in the report have been removed as recommended.
However, when introducing new function `createDistributorV2` in the `Entrypoint` contract, the event `CreateDistributorV2` from `DistributorV2Factory` has been unnecessarily removed.

Affected contracts

- DistributorV2Factory.sol#L30
- ResearchTokenFactory.sol#L69
- Entrypoint.sol#L55
- Entrypoint.sol#L67

Description

The protocol is not consistent with emitting the events. The following functions can emit events depending on the input parameter:

- `createDistributorV2`,
- `createResearchToken`,
- `createAndDistributeResearchToken`.

Additionally, the function `createAndDistributeResearchToken` emits the event with the same name as `createResearchToken` function.

Result: Inconsistent event emitting may lead to missing of some events and not responding to them.

Recommendation

- Remove conditional emitting of the events. The functions should always emit the events.
- Remove the `CreateResearchToken` from `createAndDistributeResearchToken` function and add a new one (e.g. `CreateResearchTokenAndDistributor`) that includes both, the token address and the distributor address (possibly zero-address).
- Consider naming the event in the past tense (e.g. `ResearchTokenCreated`).

References

1. G11: Code clarity

[RSPF-ed40cad-M06] Lack of *safeTransfer* for transfer of distributed tokens

MEDIUM ACKNOWLEDGED

Retest (2023-10-12)

The developers team does not plan to use SafeERC20 library at this stage, but will consider that change in future releases.

Affected contracts

- DistributorV2.sol#L109

Description

The `DistributorV2` uses the `transfer` method for moving tokens.

```
89    function claimForAccount(
90        uint256 index,
91        uint256 amount,
92        address account,
93        bytes32[] calldata merkleProof
94    ) public override {
95        if (isClaimed(index)) {
96            revert AlreadyClaimed();
97        }
98
99        // Verify the merkle proof.
100       bytes32 node = keccak256(
101           abi.encodePacked(index, amount, MERKLE_NODE_TYPE_ACCOUNT, account)
102       );
103       if (!MerkleProof.verify(merkleProof, merkleRoot, node)) {
104           revert InvalidProof();
105       }
106
107       // Mark it claimed and send the token.
108       _setClaimed(index);
109       if (!IERC20(token).transfer(account, amount)) {
110           revert TransferFailed();
111       }
112       emit ERC20ClaimForAccount(index, amount, account);
```

```
113     }
```

In the case of tokens such as USDT (Tether), it is risky due to inconsistent implementation of the `transfer` function across different token contracts. If the project allows using different tokens than `ResearchToken` or `USDC` mentioned in the documentation, it must be acknowledge that some might not return a boolean value which is now expected. Attempt to claim USDT through `claimForAccount` function would cause `revert` on every try and make the function unusable.

The impact on risk has been reduced to medium due to the fact that even though there is such a possibility. The team does not currently plan to use this type of tokens.

Result: Inability to claim tokens with non-standard `transfer` implementation (where the transfer does not return a boolean value).

Recommendation

- Replace the `transfer` function with `safeTransfer` from a reputable library such as OpenZeppelin's SafeERC20.

References

1. I2: Token

[RSPF-ed40cad-L01] The number of tokens may exceed 1 million

MINOR FIXED

Retest (2023-10-12)

The vulnerability has been removed. The cap of tokens is set to a constant value (1 million) and is enforced in `freezeMinting` and `mint` functions.

Affected contracts

- ResearchToken.sol#L41

Description

According to the documentation, each research paper should correspond to 1 million tokens, which are later distributed as intended.

" ***How does this work in practice for a researcher?***
You mint a research artifact through our website. That paper has its own supply of one million tokens, the majority going to the authors. The rest of the tokens are distributed amongst the paper's important references, as decided by the minting author, and the greater community. Our service performs this proportional allocation and posts it on the Ethereum blockchain. "

However, it is possible to set any number of tokens as `input` struct in `cap` parameter.

The `ResearchTokenInput` struct:

```
5  struct ResearchTokenInput {
6      uint256 cap;
7      string name;
8      string symbol;
9      string researchIdentifier;
10     string metadata;
11 }
```

The `initialize` function in `ResearchToken` that receives input parameters:

```
25 function initialize(
26     ResearchTokenInput memory input,
27     address minter_
28 ) public virtual initializer {
29     __ERC20Abstract_unchained(input.name, input.symbol);
30     require(minter_ != address(0), "ResearchToken: minter is zero address");
31     minter = minter_;
32     require(input.cap > 0, "ResearchToken: erc20_cap is <= 0");
33
34     erc20_cap = input.cap;
35     metadata = input.metadata;
36     researchIdentifier = input.researchIdentifier;
37 }
```

The highlighted line shows that any value is accepted for `erc20_cap`.

Vulnerable scenario

The vulnerable scenario might look like this:

① The attacker calls the `createResearchToken` function in the `ResearchTokenFactory` and transfers a cap of, for example, 100 million in the structure.
② The attacker prepares the distribution on the basis of 50% himself, 50% to the other researcher.

③ They mint 500,000 tokens for each of the addresses indicated in the distribution.
④ After obtaining funding, the attacker mints the remaining 99 million tokens, changing the proportion of receiving the prize from **50% vs. 50%** to **99.5% vs. 0.5%**.

Result of the attack: Taking over the accumulated reward by deception.

Recommendation

- If this value should be constant, define it and don't require it to be passed by the user in the `ResearchTokenFactory`.
- If the number of tokens can vary, update the documentation and remove vulnerability RSPF-ed40cad-H01, which will not allow partial minting.

References

1. SCSVS G4: Business logic

[RSPF-ed40cad-L02] The distribution amounts are not immediately verified

MINOR FIXED

Retest (2023-10-12)

The vulnerability has been removed by implementing the changes recommended in RSPF-ed40cad-H01.

Affected contracts

- ResearchToken.sol#L41

Description

The assumption resulting from the `Entrypoint` logic is to mint and distribute all tokens (equal to `erc20_cap`) immediately after the `ResearchToken` contract is deployed.

However, this requirements is verified only inside `freeze` function.

```
39    function freeze() public {
40        require(
41            tx.origin == minter,
42            "ResearchToken: Can only be frozen by minter"
43        ); // XXX tx.origin a bad choice.
44        require(!frozen, "ResearchToken: Already frozen");
```

```
45          require(
46              totalSupply() == erc20_cap,
47              "ResearchToken[ERC20]: WRONG_MINTED_TOKEN_COUNT"
48          );
49          frozen = true;
50      }
```

Using the `ResearchTokenFactory` and `DistributorV2Factory` contracts directly, the attacker can skip the `freeze` function called in `EntryPoint` contract (see RSPF-ed40cad-H01).

As a result, the attacker can promise anything in the distributor parameters because amounts will not be minted or verified. It will still not be possible to mint more tokens than `erc20_cap` (as there is validation in `mint` function), but the attacker can mint no tokens at the beginning and later mint all of them for themselves.

Attack scenario

The attackers might take the following steps in turn:

① Create a research token and distribution using the `ResearchTokenFactory` and `DistributorV2Factory` contracts directly with the following parameters:
- The `merkleDistribution` values indicating a split of 80% for other researchers and 20% for the attacker.

② After obtaining the funding, mint all tokens to the attacker address.

③ Withdraw the funding.

Result of the attack: Theft of donations from philanthropists (by fooling them) and invalid (partial) distribution of tokens.

Recommendation

- Remove vulnerability RSPF-ed40cad-H01 which will cause all tokens to be minted at one time and their sum will be checked.
  - ResearchToken.sol#L41

References

1. SCSVS G4: Business logic

[RSPF-ed40cad-L03] Lack of check for the same sizes

MINOR FIXED

Retest (2023-10-12)

The vulnerability has been removed as recommended.

Affected contracts

- Entrypoint.sol#L62

Description

The `createAndDistributeResearchToken` function is used to create a research token and distribute it via distributor contract and via direct mints of specific amounts to authors. However, the function does not check whether the size of authors' addresses list and the size of amounts list are the same.

Result: The creation and distribution of the token is reverted at the end of execution that leads to waste of gas.

Recommendation

Validate the sizes of `addresses` and `amounts` at the beginning of the function and revert if they are not the same.

References

1. G10: Gas usage & limitations

[RSPF-ed40cad-L04] Invalid parameter name

MINOR FIXED

Retest (2023-10-12)

The vulnerability has been removed as recommended.

Affected contracts

- ResearchToken.sol#L67

Description

The third parameter of `_beforeTokenTransfer` function is called `tokenId` instead if `amount`.

```
64 function _beforeTokenTransfer(
65     address from,
66     address to,
67     uint256 tokenId
68 ) internal virtual override {
```

This is probably the result of supporting the ERC1155 standard in the initial version of the code.

Result: Confuses the person reading the code.

Recommendation

Change name of the third parameter from `tokenId` to `amount`.

References

1. G11: Code clarity

6. Recommendations

[RSPF-ed40cad-R01] Consider using the latest solidity version

INFO NOT IMPLEMENTED

Retest (2023-10-12)

The recommendation has been partially implemented. The version of all contracts has been bumped up to `^0.8.19`, but the version is not clearly defined because contracts can be compiled with not earlier version than `0.8.19`.

Description

In accordance with the best security practices, it is recommended to use the latest stable versions of major Solidity releases. Very often, older versions contain bugs that have been discovered and fixed in newer versions.

Moreover, it is worth remembering that the version should be clearly specified so that all tests and compilations are performed with the same version.

Recommendation

Use a specific version of Solidity compiler (latest stable):

```
pragma solidity 0.8.21;
```

WARNING: If you want to deploy on multiple chains, be aware that some of them don't support PUSH0 opcode, which will be in your bytecode if you use solc >=0.8.20. In this situation, it is recommended to choose 0.8.19.

References

1. SCSVS G1: Architecture, design and threat modeling
2. Floating pragma SWC-103

[RSPF-ed40cad-R02] Consider changing variable name

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

A manifest is an IPFS CID that publicly indicates the location of the complete Merkle tree. Implementing a separate function to retrieve its value is unnecessary, as it incurs extra gas costs during deployment and adds superfluous code.

Recommendation

Change `_manifest` to `manifest` and make it public.

- DistributorV2.sol#L22

References

1. SCSVS G11: Code clarity

[RSPF-ed40cad-R03] Consider changing *freeze* function name

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

To better describe its purpose, consider changing *freeze* function name to *freezeMinting*.

Recommendation

Change function's name in the following place:

- ResearchToken.sol#L39

References

1. SCSVS G11: Code clarity

[RSPF-ed40cad-R04] Consider removing Context inheritance

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

The `ERC20Abstract` contract inherits from `Context` contract but never changes the logic of inherited `_msgSender` and `_msgData` functions.

Recommendation

Remove inheritance from `Context` contract.

References

1. SCSVS G11: Code clarity

[RSPF-ed40cad-R05] Remove left-over code from ERC1155

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

The `MERKLE_NODE_TYPE_ERC1155` constant is never used and is probably the left-over of ERC1155 standard that was supported in the initial version of the code.

Recommendation

Remove the constant.

References

1. SCSVS G11: Code clarity

[RSPF-ed40cad-R06] Consider using existing libraries

INFO NOT IMPLEMENTED

Retest (2023-10-12)

The recommendation has not been implemented.

Description

The repository consists of the following contracts that are part of the external libraries:

- Initializable.sol#L20
- Clones.sol#L22
- ERC20Abstract.sol#L40

The audit team has not observed any changes from the original version of those contracts.

Recommendation

Use the contracts from existing libraries.

References

1. G1: Architecture, design and threat modeling
2. SCSVS G11: Code clarity

[RSPF-ed40cad-R07] Gas optimization: Inefficient state variable increment

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

As `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables, the small amount of gas can be saved in the `ERC20Abstract` contract.

```
293 function _mint(address account, uint256 amount) internal virtual {
294     require(account != address(0), "ERC20: mint to the zero address");
295
296     _beforeTokenTransfer(address(0), account, amount);
297
298     _erc20_totalSupply += amount;
299     _erc20_balances[account] += amount;
300     emit Transfer(address(0), account, amount);
301
302     _afterTokenTransfer(address(0), account, amount);
303 }
```

Recommendation

Change the following

```
_erc20_totalSupply += amount;
_erc20_balances[account] += amount
```

to

```
_erc20_totalSupply = _erc20_totalSupply + amount;
_erc20_balances[account] = _erc20_balances[account] + amount
```

References

1. SCSVS G10: Gas usage & limitations

[RSPF-ed40cad-R08] Gas optimization: Use custom errors instead of require statements.

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

Use custom errors instead of require statements with string literals for error handling. Custom errors are not only more gas efficient due to reduced storage and execution costs but also provide clearer error messages.

Recommendation

Change the following

```
require(condition, "Error message here");
```

to

```
error CustomErrorName();
if(!condition) {
    revert CustomErrorName();
}
```

in the following places:

- DistributorV2.sol#L55-59
- DistributorV2.sol#L86
- DistributorV2.sol#L92-95
- DistributorV2.sol#L99-102
- ResearchToken.sol#L30
- ResearchToken.sol#L32
- ResearchToken.sol#L40-48
- ResearchToken.sol#L53-60
- ResearchToken.sol#L70-73
- ResearchToken.sol#L76-79
- ResearchTokenFactory.sol#L61-64

Additionally, if the recommendation **RSPF-ed40cad-R06** is not implemented, change the logic in these places:

- ERC20Abstract.sol#L234-237
- ERC20Abstract.sol#L264-265
- ERC20Abstract.sol#L270-273
- ERC20Abstract.sol#L294
- ERC20Abstract.sol#L323-324
- ERC20Abstract.sol#L345-348
- Initializable.sol#L35-38
- Clones.sol#L42
- Clones.sol#L69

References

1. SCSVS G10: Gas usage & limitations

[RSPF-ed40cad-R09] Gas optimization: Cache the array length outside a loop

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

Caching the array length outside a loop saves reading it on each iteration, as long as the array's length is not changed during the loop.

Recommendation

Cache the following lengths:

- Entrypoint.sol#L61

References

1. SCSVS G10: Gas usage & limitations

[RSPF-ed40cad-R10] Gas optimization: Redundant validation

INFO IMPLEMENTED

Retest (2023-10-12)

The recommendation has been implemented as recommended.

Description

The `_beforeTokenTransfer` function has 2 redundant checks.

```
64  function _beforeTokenTransfer(
65      address from,
66      address to,
67      uint256 tokenId
68  ) internal virtual override {
69      if (from == address(0)) {
70          require(
71              !frozen,
72              "ResearchToken: Minting is not allowed once frozen"
73          );
74      }
75      if (to == address(0)) {
76          require(
77              !frozen,
78              "ResearchToken: Burning is not allowed once frozen"
79          );
80      }
81      super._beforeTokenTransfer(from, to, tokenId);
82  }
```

The first check (for `from` parameter) is done in the `mint` function and this function is the only place where transfer is done from zero-address.

The second check (for `to` parameter) is not needed because there is no `burn` function. Moreover, users could send tokens to *0xdead* address anyway.

Recommendation

Remove redundant checks.

References

1. SCSVS G11: Code clarity

7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

8. Long-term best practices

8.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

8.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

8.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

8.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

8.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com