peppersec

PUBLIC REPORT

POA Network Token Wizard

Smart Contract Security Audit

Clarity is a rare commodity. That is why for the convenience of both the client and the reader, we have introduced a system of marking vulnerabilities and security issues we discover during our security audits.

Let's start with an ideal case. If an identified security imperfection bears no impact on the security of our client, we mark it with the No issue label.

The fixed security issues get the ✓ Fixed label that informs those reading our public report that the flaws in question should no longer be worried about.

In case a client addresses an issue in another way (e.g., by updating the information in the technical papers and specification) we put a nice Addressed tag right in front of it.

If an issue is planned to be addressed in the future, it gets the Acknowledged tag, and a client clearly sees what is yet to be done.

Although the issues marker with ✓ Fixed and Acknowledged are no threat, we still list them to provide the most detailed and up-to-date information for the client and the reader.

TABLE OF CONTENTS

Source code

Object	Location
Token Wizard App	#2840b97dea33c8cf455a67b2b9c7229e2cda1843
Auth_os	release #1.0.4

Audit methodology

The code of a smart contract has been automatically and manually scanned for known vulnerabilities and logic errors that can potentially cause security threats. The conformity of the requirements (i.e, White Paper) and practical implementation has been reviewed as well. More information on the methodology can be found **here**.

Auditors

Alexey Pertsev. **PepperSec**.

Discovered vulnerabilities

Below, you can find a table with all the discovered bugs and security issues listed.

Vulnerability description	Severity	See paragraph
No access control	Critical	Auth-os: ScriptExec
Contract does not prevent accidental Ether transferring	Major	DutchProxy
Math improvement	Minor	Auth-os: Contract
Code reuse		
Redundant code		
Code reuse		Auth-os: Abstract storage
Documentation mistype or logical flaw		DutchCrowdsale: Token
There is no “isWhitelisted” check during purchase		DutchCrowdsale: Sale
Documentation mistype		DutchCrowdsale: Admin
There is no check that _min_ token_purchase <= _max_ token_purchase		
Code reuse		
Code reuse		DutchCrowdsaleIdx
There are overflow checks		
Whitelist can be added to a non-existent tier		MintedCappedCrowdsale: SaleManager
Unnecessary functionality		ProxiesRegistry

Documentation mistype	None	Auth-os: Abstract storage
Payment can be delivered via transfer only		
Contract.Sender() can be spoofed (exec func)		
No input validation		Auth-os: ScriptExec
Token wizard app does not use authos killer feature		General issues

Math improvement

► **Severity:** Minor

Contract.sol#L494

Recommendations:

Consider using the `>=` sign instead of `>`

Status:

► Fixed – **#da5361fdc0d962e4094e62f78259578d6a15a6ef**

Code reuse

► **Severity:** Minor

The snippet of code bellow is used 16 times within Contract.sol contract:

```
// If the free-memory pointer does not point beyond the buffer's current size, update it

if lt(mload(0x40), add(0x20, add(ptr, mload(ptr)))) {

    mstore(0x40, add(0x20, add(ptr, mload(ptr))))

}
```

1. Consider taking it into a separate function.

```
function setmptr() internal pure {
    assembly {
        let ptr := add(0x20, mload(0xc0))
        if lt(mload(0x40), add(0x20, add(ptr, mload(ptr)))) {
            mstore(0x40, add(0x20, add(ptr, mload(ptr))))
        }
    }
}
```

And call it after the assembly block or alike parameter for the condition modifier which it actually is.

2. Consider calling `initialize()` instead of this code block at the authorize function.

Status:

Fixed – **#da5361fdc0d962e4094e62f78259578d6a15a6ef**

Redundant code

► **Severity:** Minor

The **Contract.sol#L333** line is a duplicate of **Contract.sol#L327**. Since the framework is under heavy development, the code can be replicated many times in the future, which is undoubtedly not a good thing.
The same can be found here and here.

Recommendations:

1. Remove it.

Status:

Fixed – **#de8aabdc8e8d6c81e7b1b2d814856488a7cd9057**

Documentation mistype

► **Severity:** None

The **AbstractStorage.sol#L452** parameter is supposed to be `n_emitted` instead of `n_paid`. **AbstractStorage.sol#L414** is the exact same thing. **AbstractStorage.sol#L64** misses `_provider` description.

Status:

Fixed – **#fca016e0aa01f177d3d2d28070d1c80ca43091eb**

Payment can be delivered via transfer only

► **Severity:** None

AbstractStorage.sol#L392 the `doPay` func implements payments via `transfer` only. Consider adding the `send` functionality. It may turn out to be extremely useful for some contracts.

Team comment:

Currently we don't have a way for users to decide what happens if a send fails, so I really think it needs to be either success or throw.

Status:

Due to the current AuthOS architecture, it takes too much to implement the `send` behavior, so all developers just should take it into account.

Code reuse

► **Severity:** Minor

AbstractStorage.sol#L574 Consider using this:

```
 function readMulti(bytes32 _exec_id, bytes32[] _locations) public view returns
(bytes32[] data_read) {

    data_read = new bytes32[](_locations.length);

    for (uint i = 0; i < _locations.length; i++) {

      data_read[i] = read(_locations[i], _exec_id); // call of `read`

    }

  }
```

instead of the actual one. Since the framework is under heavy development, code reuse is a solid approach to minimize the number of bugs.

Status:

Fixed – **#fca016e0aa01f177d3d2d28070d1c80ca43091eb**

Contract.Sender() can be spoofed (exec func)

► **Severity: None**

Due to the architectural feature, any function can be called via `AbstactStorage.exec(address sender,...)` instead of `RegistryExec.exec` or `DutchProxy`. For example, the `Token.trasfer` function uses `Contract.sender()` to get caller (considered to be `_from`), but it is just an arbitary value that the caller can send (see `AbstactStorage.exec` above). The actual authorization is implemented by `Token` contract itself - it checks whether the actual caller is `Proxy` contract.

The same is relevant for the `createInstance` function.

Recommendation:

1. All the developers who use AuthOS should be aware of the behavior of the kinfd. Consider adding this into the documentation.

Team comment:

This is intended, and is in fact used in the latest RegistryExec. Maybe a better name for the variable would be “exec_as” or something. Basically, it is part of the architecture to use a ScriptExec contract, or something similar to interface with storage, and it is up to that contract to perform input validation and provide information to storage. It’s just a separation of concerns problem, and the job here is given to ScriptExec.

Status:

There is no issue here. Just the thing that should be taken into account.

No access control

► **Severity: Critical**

There is no `onlyOwner` modifier at the **configure** function. So an attacker can use it to reconfigure app.

Recommendation:

1. Add access control for the function.

Status:

Fixed – **#d11890df8628682099af4ebc4743c8db948252bf.**

No input validation

► **Severity: None**

In contrast to other functions and **parameters**, the **configure** and **setProvider** functions do not check `provider` address.

Recommendation:

1. Consider adding some checks to keep code uniform

Team comment:

the `_provider` check is unnecessary, as any address might be a valid provider, even 0x0.

Status:

There is no issue here.

Documentation mistype or logical flaw

► **Severity:** Minor

In the **Token.sol#L198** line, it is stated “Ensures state change will ***only*** affect storage and events”. However, the actual **emitAndStore** function just checks that `emitted` and `stored` buffers are not empty (note, there is no `payment` check here). So, if a function does some unexpected manipulations with the `payment` buffer, it will not be spotted (but the comment tells us the opposite thing).

Status:

Fixed – **#9e21ef2ddc536ab9701e67db229caa8d02c2e5de**

There is no “isWhitelisted” check during purchase

► **Severity: Minor**

Sale.sol#L292 `buy` function does not check a contributor being/not being whitelisted, so `min_contribution` is set **to 0** during execution. Fortunately, that behavior is not exploitable because of zero amount exception at **line 144**.

Recommendation:

1. Consider adding an explicit check that emits readable exception message.

Status:

Team decided to leave it as is. There is no threat.

Documentation mistype

► **Severity: Minor**

Admin.sol#L378 `_max_wei_spend` parameter is used as **`_max_token_purchase`** actualy.

Recommendation:

1. Rename the parameter to avoid misunderstanding.

Status:

Fixed – **#9e21ef2ddc536ab9701e67db229caa8d02c2e5de**

There is no check that _min_token_purchase <= _max_token_purchase

► **Severity: Minor**

Admin.sol#L63 the `whitelistMulti` function does not check that `_min_token_purchase` <= `_max_token_purchase`, so they can keep any values. At this time, this cause no serious impact.

Recommendation:

1. Consider adding the check to avoid accidents.

Status:

Fixed – **#9e21ef2ddc536ab9701e67db229caa8d02c2e5de**

Code reuse

► **Severity: Minor**

Admin.sol#L322-L323 consider calling `onlyAdmin` func instead of a copy-pasting function body. Same thing **here**.

Status:

Fixed – **#9e21ef2ddc536ab9701e67db229caa8d02c2e5de**

Contract does not prevent accidental Ether transferring

► **Severity:** Major

`DutchProxy` contract has the `payble` fallback function (inherits the **Proxy.sol#L26**) *for storing refunds*. However, this may cause potential issues with the crowdsale app: according to user experience, someone can just send Ether to “crowdsale” address and lose it (the `buy` function will not be called).

Recommedation:

1. Consider using another (custom) function to refund or make the fallback function check that `msg.sender == address(app_storage)`

Status:

Fixed **#fd207315418bdc4b16482516ae6d4e6df7b0a801**

Code reuse

► **Severity: Minor**

Consider importing the `Token`, `Sale`, and `Admin` libraries instead of **copy-pasting** their functionality. After importing, it can be used the same way as `Contract` (e.g., `Contract.storing()`). So it would be:

```
Contract.set(Sale.startRate()).to(_starting_rate);
```

instead of:

```
Contract.set(startRate()).to(_starting_rate);
```

The first one is more readable and keeps code minimalistic.

Status:

Taken into account.

There are no overflow checks

► **Severity: Minor**

DutchCrowdsaleIdx.sol#L362 the `getRateAndTimeRemaining` function does not check the `_start_rate` and `_end_rate` values. So, if `_end_rate` is bigger than `_start_rate` then (at **line 377**) `uint` underflow occurs and current rate becomes huge.

Recomendations:

1. At the moment, there is no way to exploit the behavior, but this kind of check (`_start_rate` >= `_end_rate`) would be extremely useful to complicate or eliminate attacks.

Status:

Team decided to leave it as is. There is no threat.

Whitelist can be added to a non-existent tier

► **Severity: Minor**

SaleManager.sol#L145. The `whitelistMultiForTier` function does not check _tier_index, so it can be any.

Recommendation:

1. Consider adding the check to avoid accidents, i.e., `current_tier_index` <=`_tier_index` <= `last_tier_index`

Status:

Team decided to leave it as is. There is no threat.

Unnecessary functionality

► **Severity: Minor**

The new version of openzeppelin `Ownable` contract has the `renounceOwnership` function. For more information, see **here**. This function is inherited by your `ProxiesRegistry` with no indications. `renounceOwnership` seems **superfluous**.

Recomendations:

1. Consider rewrite `renounceOwnership` to empty implementation.

Status:

Fixed – **#f9d1518369d37a34bb1685416977cb891f908b1b**

Token wizard app does not use authos killer feature

► **Severity:** None

At the moment, Token and Crowdsale together is just one app. However, because of an architectural feature (AbstactStorage), they can act as separate apps that share Storage. This behavior may bring an additional layer of security.

Recommendations:

It may turn out to be a laborious task to rewrite TokenWizard, so that is up to the development team.

Status:

Taken into account.

Appendix 1. Terminology

Severity

Severity is the category that described the magnitude of an issue.

		Severity		
Impact	Major	Medium	Major	Critical
	Medium	Minor	Medium	Major
	Minor	None	Minor	Medium
		Minor	Medium	Major
		Likelihood		

Minor

Minor issues are generally subjective in their nature or potentially associated with the topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code.

The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

Medium

Medium issues are generally objective in their nature but do not represent any actual bugs or security problems.

These issues should be addressed unless there is an apparent reason not to.

Major

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise to be exploited.

If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

Critical

Critical issues are directly exploitable bugs or security vulnerabilities.

If unaddressed, these issues are likely or guaranteed to cause major problems and ultimately a full failure in the operations of the contract.

About Us

Worried about the security of your project? You're on the right way! The second step is to find a team of seasoned cybersecurity experts who will make it impenetrable. And you've just come to the right place.

PepperSec is a group of whitehat hackers seasoned by many-year experience and have a deep understanding of the modern Internet technologies. We're ready to battle for the security of your project.

LET'S KEEP IN TOUCH


peppersec.com


hello@peppersec.com


Github