Furucombo Smart Wallet and Gelato

Smart Contract Security Assessment

13.09.2021

FURUCOMBO

ABSTRACT

Dedaub (commissioned by Dinngo) has performed a security audit on a collection of new Furucombo contracts. More specifically, the following Furucombo projects were audited:

- furucombo-smart-wallet, at commit hash `3b933af5ac88b5e5b65b6d268a5338b41650dd8a` (revision at commit hash `3368d788bf2f2799fa278f4080a440272016ca15`).
- furucombo-gelato, at commit hash `00a8919fcb97c9eb45eae9cfcc500a81bf0c0dee`.

Two auditors worked over this codebase over 5 days. No high or critical severity vulnerabilities were discovered during this period.

Summary

The audited code base is of average size, at around 2 KLoC (excluding test and interface code). The audit focused on security, establishing the overall security model and its robustness, and also crypto-economic issues. Functional correctness (e.g., that the calculations are correct) was a secondary priority. Functional correctness relative to low-level calculations (including units, scaling, quantities returned from external protocols) is generally most effectively done by also complementing with thorough testing in addition to human auditing.

The contracts in scope for this audit make heavy use of dapphub's DSProxy architecture, used also by DeFi Saver for a similar purpose. The user's DSProxy delegatecalls into several of the audited contracts (i.e. ones under `furucombo-smart-wallet`) allowing the state changes made by these contracts to only affect the user's DSProxy instance. The contracts under the `furucombo-gelato` repo allow a user's DSProxy to set up Tasks that can be executed periodically by the Gelato contract.

furucombo-smart-wallet

The contracts in this repo define a set of Action contracts that can be used to perform state changes on a user's `DSProxy` if that `DSProxy` instance delegatecalls into them.

The entry-point and most complex of these actions is the `TaskExecutor` contract. It can be used to define complex actions by performing a series of external calls. These external calls can either be calls to external contracts (i.e. Tokens or other DeFi services) or `delegatecalls` into other actions (several of which are implemented already and part of this audit's scope). The series of calls for each execution of the `TaskExecutor` can affect one another, with `localStack`, a 256 length 32-byte-element array being used to store the return data of a call that is to be referenced by a following call. To support this, a 32-byte `config` value is supplied for each external call (in addition with the call's target and calldata).

This config value contains tightly packed information about the call's interaction with the `localStack`. It specifies whether its return data will be read (referenced) by a future call (in that case the entire return data is copied to the `localStack`) and if some words from its calldata need to be overwritten by words read from the localStack.

In order for the system to function correctly the invoker of the `TaskExecutor` needs to provide correct config values for each sub-call. This would ensure the mapping of values to and from the localStack is performed correctly. Although some sanity checking is performed (in cases of calls with their return data copied to the `localStack`, its length is verified to be the same as a value written in the `config` value), there is plenty of room for error when creating the config values. This is made more challenging due to the type agnostic nature of the localStack buffer. The types of the values written to it are "forgotten", leaving it up to the provider of the config values to ensure no type errors take place. This type agnostic nature however, allows it to take advantage of solidity's ABI encoding and support dynamic arrays and other complex data structures without having to implement that support.

It will be therefore important that each created task be audited to the same level of detail as if these were coded by hand.

furucombo-gelato

The contracts in the furucombo-gelato repo implement a system allowing users to set up specific tasks that can be periodically executed by third-parties through their DSProxies.

The entry-point of this system is the `FuruGelato` contract, implementing (including its superclasses) the bulk of the system's logic. A non-blacklisted user can create/register a new Task through a number of whitelisted Resolvers, which check the validity of the new Task and implements the conditions that need to be met for the Task to be executed. A Task is identified by the keccak256 hash of the 3-tuple
(DSProxy addr, Resolver addr, ExecutionData), allowing a single user to register multiple individual Tasks. Each registered Task can be executed by the Gelato contract, given the conditions specified by the Resolver are met.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description

CRITICAL	Can be profitably exploited by any knowledgeable third party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: 01) User or system funds can be lost when third party systems misbehave. 02) DoS, under specific conditions. 03) Part of the functionality becomes unusable due to programming error.
LOW	Examples: 01) Breaking important system invariants, but without apparent consequences. 02) Buggy functionality for trusted users where a workaround exists. 03) Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed", by the client, or "resolved", per the auditors.

CRITICAL SEVERITY

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

ID	Description	STATUS
M1	Possible corruption in Task Executor	**ACKNOWLEDGED**
As we mentioned in the Summary section, the creator of the TaskExecutor.batchExec() invocation payload needs to keep a precise record of each call's calldata and returndata in order to ensure that the created `localStack` will be correct. To ensure that a call's returndata is what was expected its length has to be the same with a value written to the `config` value. Assuming proper ABI encoding of the returndata, this check is enough to ensure proper returndata format for calls that return up to 1 dynamic array. However, for calls that return 2 or more dynamic arrays, the total returndata size can be the same but the lengths of the returned arrays can be different. In this case the computations producing the `config` values can be wrong, leading to data corruption.		

LOW SEVERITY:

ID	Description	STATUS
L1	Proliferous use of weak blacklists	**ACKNOWLEDGED**
Furucombo Gelato makes use of a number of blacklists including: - Who can create a new task - What task can be created It is however trivial for any user to get around this blacklisting style. For instance, in the case of a task, one can simply add some additional calldata which does not affect the semantics of the task. Therefore, if there is a reason to blacklist users or tasks, a stronger mechanism needs to be designed.		
L2	`delegateCallOnly` methods not properly guarded in Actions	**CLOSED**
In TaskExecutor the `delegateCallOnly()` modifier is defined to ensure that the `batchExec()` method is only called via delegate call, as intended by the deployers. This can be reused by the other Actions as well, to make sure that they are not misused.		

OTHER/ ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend addressing them.

ID	Description	STATUS
A1	Floating pragma	**CLOSED**
	The floating pragma “`pragma solidity ^0.6.0;`” is used in most contracts, allowing them to be compiled with the 0.6.0 - 0.6.12 versions of the Solidity compiler. Although the differences between these versions are small, floating pragmas should be avoided and the pragma should be fixed to the version that will be used for the contracts’ deployment.	
A2	Compiler known issues	**INFO**
	The contracts were compiled with the Solidity compiler 0.6.12 which, at the time of writing, has multiple issues related to memory arrays. Since furrucombo-smart-wallet makes heavy use of memory arrays, and sending and receiving these to third party contracts, it is worth considering switching to a newer version of the Solidity compiler.	

DISCLAIMER

ABOUT DEDAUB

Dedaub offers technology and auditing services for smart contract security. The founders, Neville Grech and Yannis Smaragdakis, are top researchers in program analysis. Dedaub's smart contract technology is demonstrated in the contract-library.com service, which decompiles and performs security analyses on the full Ethereum blockchain.