Matchpool GUP Token Audit

OPENZEPPELIN SECURITY | MARCH 31, 2017

Security Audits

The Matchpool team asked us to review and audit their new GUP token code. We looked at their contracts and now publish our results.

The audited contracts can be found on this URL. The main contracts are GUPToken and Contribution. The audited code was timestamped with the hash d5b9c96c234495c9ae53e64571ab25b62dcc11bae8b282b784624c65aa5ef399.

Here's our assessment and recommendations, in order of importance:

Update: Since publishing this post, Matchpool team has implemented most recommendations found in this report.

Severe

We haven't found any severe security problems with the code.

Potential problems

Use safe math

There are many unchecked math operations in the code. For example, all additions and differences in StandardToken are not checked for overflows and underflows. It's always better to be safe and perform checked operations. Even though the code uses a SafeMath class, it only supports safeMul and safeDiv, and no safe addition or difference. Consider using a complete safe math library, or performing pre-condition checks on any math operation.

when a previously correct component gets broken based on a recent change. Consider adding tests to catch regression errors when updating the contract's code and to further check correct functioning of contracts.

Update: Tests added by Matchpool team here.

Timestamp usage

There's a problem with using timestamps and **now** (alias for **block.timestamp**) for contract logic, based on the fact that miners can perform some manipulation. In general, it's better not to rely on timestamps for contract logic. The solutions is to use **block.number** instead, and approximate dates with expected block heights and time periods with expected block amounts.

The **GUPToken** and **Contribution** contracts use timestamps at several points. The risk of miner manipulation, though, is really low. The potential damage is also limited: miners could only slightly manipulate when minting ends, and crowdfunding price steps. This probably won't affect the functioning of the contract. We recommend the team to consider the potential risk of this manipulation and switch to **block.number** if necessary.

For more info on this topic, see this stack exchange question.

Warnings

Use of send

Use of **send** is always risky and should be analyzed in detail. Two occurrences found in line 341 and line 374.

- Always check send return value: OK.
- Consider calling send at the end of the function: Warning. send call in processPurchase could be moved further down. Consider changing send calls to be the last thing the function does, as it's an external interaction.
- Favor pull payments over push payments: Warning. All occurrences of send are push payments. Although we couldn't find any attack vectors on this contract, consider using OpenZeppelin's PullPayment contract to implement pull payments.

Formal security audits are not enough to be safe. We recommend implementing an automated contract-based bug bounty and setting a period of time where security researchers from around the globe can try to break the contract's invariants. For more info on how to implement automated bug bounties with OpenZeppelin, see this guide.

Use latest version of Solidity

Current code is written for an old version of solc (0.4.6). We recommend changing the solidity version pragma for the latest version (`pragma solidity ^0.4.10;`) to enforce latest compiler version to be used.

edit: fixed by Matchpool team in the latest version of the code

Use OpenZeppelin's StandardToken, Ownable and SafeMath contracts

Rewriting the same StandardToken contract for every project is prone to bringing unexpected problems. That's the reason why we created OpenZeppelin as an open-source framework of reusable and secure smart contracts. Consider using OpenZeppelin's StandardToken instead of re-implementing it. Same with OpenZeppelin's SafeMath and Ownable, which could be used to replace repeated code.

Code modularity

All contracts appear in a single large file. Simpler code means easier audits, and better understanding of what each component does. Consider separating current code into small files, small contracts, and small functions. If you can separate a contract into many independent functionalities you should probably do it.

edit: fixed by Matchpool team in the latest version of the code

Additional Information and Notes

- GUPToken is ERC20 compliant.
- Matchpool address gets 40% of tokens via **ALLOC_ILLIQUID_TEAM**, **ALLOC_LIQUID_TEA**, **ALLOC_BOUNTIES**, **ALLOC_NEW_USERS**.
- Good work on using an emergency stop mechanism.

- Good work limiting the amount of funds collected.
- The safeguard fallback function in lines 122–124 is not needed if using solidity >0.4.0, as the **payable** keyword was introduced.
- Some modifiers use throw and some fail silently. Consider changing all to throw, to fail early and loudly.
- **publicStartTime** and **privateStartTime** are in the past. **publicStartTime** is Sat, 25 Mar 2017 13:00:00 GMT and **privateStartTime** is Sat, 25 Mar 2017 09:00:00 GMT.
- Comment in line 139 has a typo. Should read: “Can only be called by crowdfund contract before the end time.”
- Return value of **makeLiquid** function is always true, and thus useless. Consider removing the return bool value. Same for **createIlliquidToken**, and **createToken**.
- Comment in line 82 reads “Time in seconds no more tokens can be created”, and it should say “Timestamp after which no more tokens can be created” or something similar.
- Good job naming constants in ALL_CAPS_AND_UNDERSCORES.

Conclusions

No severe security issues were found. Some changes were recommended to follow best practices and reduce potential attack surface.

Update: Since publishing this post, Matchpool team has implemented most recommendations found in this report.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the GUP token. We have not reviewed the related MatchPool project. The above should not be construed as investment advice or an offering of GUP tokens. For general information about smart contract security, check out our thoughts here.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits