EtherCamp's decentralized startup team public code audit

OPENZEPPELIN SECURITY | DECEMBER 18, 2016 Security Audits

Following our audits of the HackerGold (HKG) token and the ProjectKudos contract, we've been asked by the EtherCamp team to review their decentralized startup team contract.

The audited contracts can be found in the master branch of their GitHub repo, specifically, commit 6d4097a08669c2520e0d5bab317b60f1d13df44e. Main contract file is DSTContract.sol.

Severe

collectedEther subject to manipulation

Line 159 of DSTContract.sol isn't considering the fact that `collectedEther` can actually need to be increased by a smaller amount than `msg.value`. This is due to the change return mechanism in lines 144–151. If that branch of code is executed, collected ether in that transaction should be smaller than `msg.value` (should be `msg.value - retEther`).

This allows an attacker to send a transaction with a big ether amount, receive the extra value back, and make `collectedEther` to be much bigger than the contract's real `balance`. If the ether value sent is big enough, the attacker can then circumvent the 20% limit on ether proposals because `collectedEther` can be made, for example, 5 times the contract's `balance`.

Fix this problem by changing that line to:

EDIT: Commit with fix: https://github.com/ether-camp/virtual-accelerator/commit/eaa09a20d29696927bbe11bac6dcf0ac723b8a05

issuePreferedTokens vulnerable to multiple calls

If issuePrefered is called twice, the second time line 208 is executed, the old value of `allowed[this][virtualExchangeAddress]` will be overwritten.

EDIT: Commit with fix: https://github.com/ether-camp/virtual-accelerator/commit/deae2e7147f91120dcc8055545a201083d7aae86

Potential problems

Use safe math

There are many unchecked math operations in the code. We couldn't find any related attack vectors on the DSTContract, but it's always better to be safe and perform checked operations. Consider using a safe math library, or performing pre-condition checks on any math operation.

The fact that HKG supply is limited to 4,000,000 ether (and thus at most 800,000,000 HKG assuming all tokens are created at the best possible price) helps prevent possible overflows.

Use latest version of Solidity

Current code is written for old versions of solc (0.4.2). With the latest storage overwriting vulnerability found on versions of solc prior to 0.4.4, there's a risk this code can be compiled with vulnerable versions. We recommend changing the solidity version pragma for the latest version (`pragma solidity ^0.4.6;`) to enforce latest compiler version to be used.

Timestamp usage

The code uses timestamps for contract logic. 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.

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

Be careful with small transactions

Transactions executing the fallback function with amounts smaller than `1 finney` will assign zero tokens to the sender. Same problem happened with HackerGold.sol and was fixed in that case. We recommend doing the same for `DSTContract.sol`. Same for the reverse operation.

General code quality

Code quality of the DSTContract is low, which made auditing it hard. We recommend a refactor to improve code quality (more recommendations given next). Simpler code makes functionality more apparent and reduces attack surface. Given the high degree of test coverage, making changes to improve code quality will have a very low risk of introducing regressions.

Code too long and complex

DSTContract code is too long and complex. Security comes from a match between developer intention and what the code actually does. This is very hard to verify, especially if the code is huge and messy. That's why it's important to write simple and modular code. 700+ lines-of-code files like DSTContract.sol are not recommended.

The contract could be modularized in a parent contract that handles `executive` auth and `ImpeachmentProposals`, and the rest of the code, at least.

Remove duplicate code

Duplicate code makes it harder to understand the code's intention and thus, auditing the code correctly. It also increases the risk of introducing hidden bugs when modifying one of the copies of some code and not the others. We recommend the following to remove duplicate code:

- `submitEtherProposal` and `submitHKGProposal` share a lot of code, which could be extracted into a generic function.
- The first check inside the fallback function could be removed by adding the `onlyAfterEnd` modifier.

- `isExistByString` could be implemented by calling `isExistByBytes`.
- This check in `issuePreferredTokens` is a duplicate of the `onlyIfAbleToIssueTokens` function modifier.

Remove unnecessary code

- convert function could be replaced by simply casting strings to bytes32.
- The two separate transfers in the buy function of VirtualExchange.sol could be replaced by a single call of `hackerGold.transferFrom(msg.sender, dstContract.getAddress())`. Same idea can be applied to these lines in DSTContract.sol.
- selfAddress in line 33 of DSTContract is unnecessary. Use `this`.
- Some constant getter functions can be removed if variables are made public.

Use of send

Use of `send` is always risky and should be analyzed in detail. Three occurrences found in line 150, line 495, and line 520 of DSTContract.sol.
– Always check send return value: Line 150: OK, line 495: warning, return value not checked! line 520: warning, return value not checked.
– Consider calling send at the end of the function: Line 150: warning, code executed after send. Line 495: warning, code executed after send. Line 520: warning, code executed after send.
– Favor pull payments over push payments: Warning. All lines use push payments. Consider using pull payments.

Lines 495 and 520 are guarded by the `onlyExecutive` function modifier, so only the executive address can cause problems. In line 150, we haven't identified any exploitable attacks that abuse these anti-patterns.

Warnings

Usage of magic constants

There are several magic constants in the contract code. Some examples are:

- https://github.com/ether-camp/virtual-accelerator/blob/6d4097a08669c2520e0d5bab317b60f1d13df44e/contracts/DSTContract.sol#L393-L394
- https://github.com/ether-camp/virtual-accelerator/blob/6d4097a08669c2520e0d5bab317b60f1d13df44e/contracts/DSTContract.sol#L405
- https://github.com/ether-camp/virtual-accelerator/blob/6d4097a08669c2520e0d5bab317b60f1d13df44e/contracts/DSTContract.sol#L140

Use of magic constants reduces code readability and makes it harder to understand code intention. We recommend extracting magic constants into contract constants.

Additional Information and notes

- Comments in lines 207 and 391 of DSTContract.sol have typos, saying “ammount” instead of “amount”.
- Comment in line 38 of DSTContract.sol has typo, should say “traded” instead of “threaded”.
- Comments in line 34 and line 71 of VirtualExchange.sol has a typo. Should read “acquisition” instead of “aquasition”.
- Reduce the number of external calls by extracting the result of `dstContract.getDSTSymbolBytes()` and reusing in https://github.com/ether-camp/virtual-accelerator/blob/6d4097a08669c2520e0d5bab317b60f1d13df44e/contracts/VirtualExchange.sol#L44 and https://github.com/ether-camp/virtual-accelerator/blob/6d4097a08669c2520e0d5bab317b60f1d13df44e/contracts/VirtualExchange.sol#L50.
- Comment in line 65 of VirtualExchange.sol has a typo: should read “specific” instead of “speciphic”.
- Comment for DSTContract constructor is confusing. We couldn’t make any sense of it.

Conclusions

EDIT: EtherCamp followed many of our recommendations and fixed the code based on our comments, on these commits.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to decentralized startup team contract. We have not reviewed the related Virtual Accelerator project. The above should not be construed as investment advice or an offering of 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