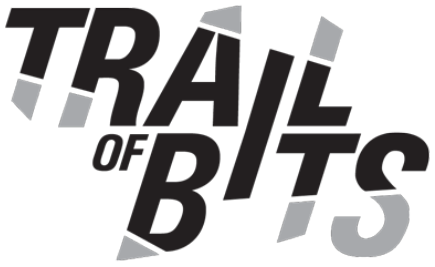

MYKEY

Security Assessment

September 14, 2020

Prepared For:
Ricky Shi | *MYKEY*
ricky@mykey.org

Prepared By:
Sam Moelius | *Trail of Bits*
sam.moelius@trailofbits.com

Sam Sun | *Trail of Bits*
sam.sun@trailofbits.com

Changelog:

July 24, 2020:	Initial report delivered
August 18, 2020:	Added Appendix E. Fix Log
August 26, 2020:	Copyedited
September 14, 2020:	Added discussion of commit `ec68db2` to Append E

Executive Summary

From July 6 through July 24, 2020, MYKEY engaged Trail of Bits to review the security of the MYKEY ETH and EOS contracts. Trail of Bits conducted this assessment over the course of four person-weeks with two engineers working from various zip archives provided via email and Slack.

During the first week, we verified that we could compile both the ETH and EOS contracts. We also verified that the ETH unit tests pass, and collected their code coverage using `solidity-coverage`. We ran Slither (Trail of Bits' Solidity source code analyzer) over the ETH contracts. Finally, we began manual review of the ETH contracts.

During the second week, we finished manual review of the ETH contracts, and began manual review of the EOS contracts. We built the EOS contracts with all C++ warnings enabled. We deployed them locally and interacted with them using the provided test scripts. Finally, we converted them to C using `wasm2c` and fuzzed them using AFL.

Our efforts resulted in 12 findings ranging from medium to informational severity. The one medium-severity finding concerns how expired or soon-to-be-expired backups could be reintroduced (TOB-MYKEY-005). The one low-severity finding concerns how proposed method IDs are computed (TOB-MYKEY-006). The remaining 10 findings are informational.

In addition to the 12 findings, we prepared multiple appendices. Notably, Appendix C contains findings that do not have immediate or obvious security implications, while Appendix D describes our threat model for the EOS and ETH contracts.

Note that we were not able to diagnose all fuzzing results prior to issuing this report. We will update MYKEY via Slack following further investigation of those results.

Our main recommendation is that additional unit tests be developed for both the ETH and EOS contracts. The ETH contracts' "happy" (i.e., successful) paths are tested quite extensively. However, they would benefit from additional tests of their "sad" (i.e., failing) paths. The EOS contracts would benefit from unit tests that could be run locally, following each code modification, for example. Something analogous to "`truffle test`" would be ideal. A comprehensive set of unit tests that can be run locally will help expose errors and protect against regressions.

A secondary recommendation concerns the use of `this` calls in the ETH logic contracts. The contracts protect functions intended for `this` calls with the `allowSelfCallsOnly` modifier. However, our concern is that an attacker could benefit from making a `this` call to a publicly exposed function that is not protected by the `allowSelfCallsOnly` modifier. To address this concern, we recommend adopting one of the following strategies:

- Explicitly whitelist all `this` calls. Specifically, for each logic contract, maintain a list of function identifiers allowed for `this` calls. Before making a `this` call, ensure it involves a function whose identifier is on the list. Note that the `ProposalLogic` contract already employs such a mechanism for proposed method IDs.
- Use build scripts to ensure new public functions are not introduced. Specifically, for each logic contract, write a build script to verify that its ABI includes only public functions from a known, good ABI. Consider the presence of any additional public functions a build failure. If a code modification introduces a new public function to a contract's ABI, consider whether the function should be made internal, or whether the known, good ABI should be updated.

Adopting one of these two strategies will help protect your users from bugs like the one found in `executeProposal` just before this assessment.

Finally, the informational severity of some of the findings is subject to the correct usage of an API (TOB-MYKEY-003, TOB-MYKEY-004, TOB-MYKEY-009). We could not verify that these APIs are used correctly since we did not have access to MYKEY's backend server code. Consider obtaining a security review of MYKEY's backend server code.

Project Dashboard

Application Summary

Name	MYKEY

Version	`MYKEY-EOS-Contracts-ToB-review-20200703.zip`
	`MYKEY-EOS-Contracts-ToB-review-20200714.zip`
	`MYKEY-ETH-Contracts-ToB-review-20200722.zip`

Type	Solidity, C++
Platforms	ETH, EOS

Engagement Summary

Dates	July 6 through July 24, 2020
Method	Whitebox
Consultants Engaged	2
Level of Effort	4 person-weeks

Vulnerability Summary

Total High-Severity Issues	0	
Total Medium-Severity Issues	1	■
Total Low-Severity Issues	1	■
Total Informational-Severity Issues	10	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Total	12	

Category Breakdown

Auditing and Logging	1	■
Data Validation	4	■ ■ ■ ■
Denial of Service	1	■
Patching	3	■ ■ ■
Timing	1	■
Undefined Behavior	2	■ ■
Total	12	

Code Maturity Evaluation

In the table below, we review the maturity of the codebase and the likelihood of future issues. In each area of control, we rate the maturity from strong to weak, or missing, and give a brief explanation of our reasoning.

Category Name	Description
Access Controls	**Satisfactory.** Though we found no issues related to the use of `this` calls in the ETH logic contracts, we recommend additional mitigations to protect against their misuse (see Executive Summary).
Arithmetic	**Satisfactory.** We found one issue regarding the handling of backup expiries. No other significant arithmetic issues were found.
Assembly Use	**Satisfactory.** An issue was found in the assembly code that computes proposed method IDs. No other significant assembly use issues were found.
Centralization	**Satisfactory.** The `LogicManager` contract is owned by a multi-sig contract. The `AccountCreator` contract is multi-owned, making it more susceptible to takeover. However, this would only affect newly deployed accounts, not existing ones.
Contract Upgradeability	**Strong.** Applies to the `AccountCreator`'s ability to update its `LogicManager`, and the `LogicManager`'s ability to update its individual logic contracts. No issues were found regarding either.
Function Composition	**Moderate.** We found the extensive use of `this` calls in the ETH contracts, and the use of transaction forwarding in the EOS contracts, to be confusing. We recommend reducing or eliminating these practices where possible.
Front-Running	**Strong.** No front-running possibilities were noted.
Monitoring	**Not Reviewed.**
Specification	**Strong.** A specification accompanies the implementation. No significant deviations were found between the specification and the implementation.
Testing & Verification	**Weak.** Few of the ETH contract's unsuccessful paths are tested. The file `LogicManager.sol` is largely untested. The EOS contracts do not have tests that can be run locally.

Engagement Goals

The engagement was scoped to assess the security of MYKEY's ETH and EOS contracts.

Specifically, we sought to answer the following questions:

- Can an attacker add themselves as a backup to an account?
- Can an attacker change an account's admin key?
- Can an attacker add an operation key to an account?
- Can an attacker write fake or dirty data into an account's storage?
- Can an attacker overwrite an account's existing storage?
- If a proposal is canceled/rejected, is it possible for an account to overlook the cancellation/rejection?
- Can an attacker reduce the pending time for an event?
- Can an attacker access another user's assets?

Coverage

ETH `Account` contract. Statically analyzed using Slither. Unit tests verified to pass and reviewed for code coverage. Manually reviewed.

ETH `AccountCreator` contract. Statically analyzed using Slither. Unit tests verified to pass and reviewed for code coverage. Manually reviewed.

ETH `AccountProxy` contract. Statically analyzed using Slither. Unit tests verified to pass and reviewed for code coverage. Manually reviewed.

ETH `AccountStorage` contract. Statically analyzed using Slither. Unit tests verified to pass and reviewed for code coverage. Manually reviewed.

ETH logic contracts (`BaseLogic`, `AccountBaseLogic`, `AccountLogic`, `CommonStaticLogic`, `DappLogic`, `DualsigsLogic`, `ProposalLogic`, `TransferLogic`). Statically analyzed using Slither. Unit tests verified to pass and reviewed for code coverage. Wrote additional unit tests. Manually reviewed.

ETH `LogicManager` contract. Statically analyzed using Slither. Unit tests verified to pass and reviewed for code coverage (see TOB-MYKEY-002). Manually reviewed.

EOS `AccountLogic` contract. Built with all C++ warnings enabled. Locally deployed and tested using provided test scripts. Fuzzed using AFL following conversion to C using `wasm2c`. Manually reviewed.

EOS `AccountMgr` contract. Built with all C++ warnings enabled. Locally deployed and tested using provided test scripts. Fuzzed using AFL following conversion to C using `wasm2c`. Manually reviewed.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

☐ **Measure the gas savings from optimizations, and carefully weigh that against the possibility of an optimization-related bug.** Also, consider using the binary version of `solc`. These steps will help prevent a bug from entering the codebase. TOB-MYKEY-001

☐ **Add unit tests. Ideally, there will be at least one test for each of a function's "happy" (i.e., successful) paths, and at least one test for each of a function's "sad" (i.e., failing) paths. Write unit tests for the `LogicManager` contract. Write unit tests for the EOS contracts that can be run locally. Something analogous to "`truffle test`" would be ideal.** A comprehensive set of unit tests that can be run locally will help expose errors, protect against regressions, and provide a sort of documentation to users. TOB-MYKEY-002

☐ **Consider calling `init` from `Account` and `AccountProxy`'s constructors.** This will prevent a race condition from allowing an attacker to set an account's `manager` field. TOB-MYKEY-003

☐ **Change the uses of `abi.encodePacked` to `abi.encode` in `createCounterfactualAccount` and `getCounterfactualAccountAddress`.** The changes to `createCounterfactualAccount` will eliminate potential collisions in calls to `create2`. TOB-MYKEY-004

☐ **Have `removeBackup` validate that the backup is eligible for removal. This consists of reverting when `_backup` is currently expiring or has already expired, and when the backup is not yet effective.** This will prevent `removeBackup` from incorrectly modifying backups. TOB-MYKEY-005

☐ **Replace the existing use of `mload` with `mload(add(add(_b, /*32+4+32*/ 68), mload(add(_b, /*32+4+32+32*/ 100))))`.** This will ensure that proposed method IDs are computed correctly. TOB-MYKEY-006

☐ **Load the address outside of assembly and ensure that storage variables are not reordered, only appended to.** This will ensure that `AccountProxy` continues to work correctly as it is updated. TOB-MYKEY-007

☐ **Update the code so that it builds with `eosio.cdt` 1.7.0.** Using out-of-date dependencies could mean critical bug fixes are missed. TOB-MYKEY-008

☐ **Clearly indicate the potential to receive incorrect information when relying on the initialization events for the various logic contracts.** Also consider indexing the `wallet` parameter in `BaseLogic`'s `LogicInitialized` event. This will help ensure that users do not unduly trust incorrect or faulty data. TOB-MYKEY-009

☐ **Update the `base58` decoder to the latest version.** This will allow MYKEY contracts to benefit from changes implemented in that version. TOB-MYKEY-010

☐ **Update the structure to include the `proposer` parameter.** This will eliminate a possible source of undefined behavior. TOB-MYKEY-011

☐ **Provide a way for account owners to cancel dual-signed proposals, or ensure that all dual-signed proposals are cleared when the admin key is changed.** This will close a gap in the MYKEY contracts' functionality. TOB-MYKEY-012

Long term

☐ **Monitor the development and adoption of Solidity compiler optimizations to assess their maturity.** This will help prevent a potentially dangerous feature from becoming enabled until it is ready for production use. TOB-MYKEY-001

☐ **As new functions are added to the codebase, ensure unit tests are written for them.** This will help ensure those functions do not introduce bugs into the codebase. TOB-MYKEY-002

☐ **As new contracts are added to the codebase, ensure that they are properly initialized in their constructors.** Doing so will prevent race conditions from entering the codebase. TOB-MYKEY-003

☐ **Prefer `abi.encode` over `abi.encodePacked` whenever multiple dynamic arguments are involved.** This will prevent more of these collisions from entering the codebase. TOB-MYKEY-004

☐ **Add tests for failing conditions.** A more comprehensive set of unit tests could have exposed this bug (see also TOB-MYKEY-002). TOB-MYKEY-005

☐ **Consider using `abi.decode()` when parsing message data.** This would eliminate the need for the assembly code that is now used. TOB-MYKEY-006

☐ **Keep in mind how Solidity lays out storage variables when writing contracts using `delegatecall`.** This will help prevent future bugs involving storage layout. TOB-MYKEY-007

☐ **Regularly monitor the `eosio.cdt` repository for updates.** If an update is for a vulnerability, consider alerting your users and redeploying your contracts so your users do not rely on vulnerable code. TOB-MYKEY-008

☐ **Monitor changes to the `base58` decoder so new patches can be merged.** This will allow MYKEY contracts to benefit from future changes to this code. TOB-MYKEY-010

☐ **Write tests to ensure that all actions can be properly unpacked.** This will help catch similar future bugs. TOB-MYKEY-011

☐ **Consider whether all workflows are implemented properly.** Diagrams may help to visualize state transitions and thereby identify gaps in functionality. TOB-MYKEY-012

Findings Summary

#	Title	Type	Severity
1	Solidity compiler optimizations can be dangerous	Undefined Behavior	Informational
2	Insufficient unit tests	Patching	Informational
3	Race condition on `Account.init`	Timing	Informational
4	Use of `abi.encodePacked` could result in collisions	Data Validation	Informational
5	Function `removeBackup` does not sufficiently validate backups	Data Validation	Medium
6	Proposed method ID is calculated incorrectly	Data Validation	Low
7	Unsafe storage load	Data Validation	Informational
8	EOS contracts do not build with the latest developer tools	Patching	Informational
9	`LogicInitialized` events cannot be trusted	Auditing and Logging	Informational
10	`DecodeBase58` implementation is not up to date	Patching	Informational
11	Incorrect definition for `approveprop_action`	Undefined Behavior	Informational
12	Dual-signed proposals cannot be canceled	Denial of Service	Informational

1. Solidity compiler optimizations can be dangerous

Severity: Informational
Type: Undefined Behavior
Target: `truffle-config.js`

Difficulty: Low
Finding ID: TOB-MYKEY-001

Description
MYKEY has enabled optional compiler optimizations in Solidity.

There have been several bugs with security implications related to optimizations. Moreover, optimizations are actively being developed. Solidity compiler optimizations are disabled by default. It is unclear how many contracts in the wild actually use them, so it is unclear how well they are tested and exercised.

High-severity security issues due to optimization bugs have occurred in the past. A high-severity bug in the Emscripten-generated `solc-js` compiler used by Truffle and Remix persisted until late 2018. The fix for this bug was not reported in the Solidity `CHANGELOG`. Another high-severity optimization bug resulting in incorrect bit shift results was patched in Solidity 0.5.6.

A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe. Moreover, the Common Subexpression Elimination (CSE) optimization procedure is "implemented in a very fragile manner, with manual access to indexes, multiple structures with almost identical behavior, and up to four levels of conditional nesting in the same function." Similar code in other large projects has resulted in bugs.

There are likely latent bugs related to optimization, and/or new bugs that will be introduced due to future optimizations.

Exploit Scenario
A latent or future bug in Solidity compiler optimizations—or in the Emscripten transpilation to `solc-js`—causes a security vulnerability in MYKEY contracts.

Recommendation
Short term, measure the gas savings from optimizations, and carefully weigh that against the possibility of an optimization-related bug. Also, consider using the binary version of `solc`. These steps will help prevent a bug from entering the codebase.

Long term, monitor the development and adoption of Solidity compiler optimizations to assess their maturity. This will help prevent a potentially dangerous feature from becoming enabled until it is ready for production use.

2. Insufficient unit tests

Severity: Informational
Type: Patching
Target: Various

Difficulty: Low
Finding ID: TOB-MYKEY-002

Description
MYKEY's ETH contracts would benefit from additional unit tests, particularly for failing conditions. MYKEY's EOS contracts would benefit from unit tests that could be run locally.

The existing ETH unit tests cover the code's "happy" (i.e., successful) paths rather extensively. However, many of its "sad" (i.e., failing) paths are not covered.

For example, the contracts feature 120 `require` statements:

- 22 of these are not covered by unit tests.[1]
- The 98 that are covered are never tested to fail.

In addition, the file `LogicManager.sol` is largely untested. Only 23% of the file's statements are covered by unit tests.

MYKEY provides a set of scripts to interact with an existing EOS contract deployment on the Kylin testnet. However, the contracts would benefit from tests that could be run locally, following each code modification, for example.

Unit tests help expose errors, and provide a sort of documentation to readers of the code. Moreover, they exercise code in a more systematic way than any human can. In this way, they help protect against regressions. For these reasons, it is important that MYKEY have a comprehensive set of unit tests.

Exploit Scenario
Eve exploits a flaw in the MYKEY protocol. The flaw would likely have been revealed through unit tests.

Recommendation
Short term:

- Add unit tests. Ideally, there will be at least one test for each of a function's "happy" (i.e., successful) paths, and at least one test for each of a function's "sad" (i.e., failing) paths.
- Write unit tests for the `LogicManager` contract.

[1] In fairness, some of these `require` statements appear in parts of `SafeMath.sol` and `RLPReader.sol` that are unused by MYKEY.

- Write unit tests for the EOS contracts that can be run locally. Something analogous to "`truffle test`" would be ideal.

A comprehensive set of unit tests that can be run locally will help expose errors, protect against regressions, and provide a sort of documentation to users.

Long term, as new functions are added to the codebase, ensure unit tests are written for them. This will help ensure those functions do not introduce bugs into the codebase.

3. Race condition on `Account.init`

Severity: Informational
Type: Timing
Target: `contracts/Account.sol`

Difficulty: Undetermined
Finding ID: TOB-MYKEY-003

Description
The function `Account.init` is called to set an account's `manager` field. Since anyone can call this function, anyone can set an account's `manager` field.

The entire `init` function appears in Figure 3.1. Note that the passed-in value `_manager` is only minimally validated: It must be non-null, and it must affirmatively respond to certain `isAuthorized` calls.

```
    function init(address _manager, address _accountStorage, address[] calldata _logics,
address[] calldata _keys, address[] calldata _backups)
        external
    {
        require(manager == address(0), "Account: account already initialized");
        require(_manager != address(0) && _accountStorage != address(0), "Account: address
is null");
        manager = _manager;

        for (uint i = 0; i < _logics.length; i++) {
            address logic = _logics[i];
            require(LogicManager(manager).isAuthorized(logic), "must be authorized
logic");

            BaseLogic(logic).initAccount(this);
        }

        AccountStorage(_accountStorage).initAccount(this, _keys, _backups);

        emit AccountInit(address(this));
    }
```

Figure 3.1: `contracts/Account.sol#L30-L47`.

An account's logic manager determines the logic contracts that have access to the account (see Figure 3.2). Given that the logic manager has such a profound effect on an account, the field should be set in `Account`'s constructor.

```
    modifier allowAuthorizedLogicContractsCallsOnly {
        require(LogicManager(manager).isAuthorized(msg.sender), "not an authorized
logic");
        _;
    }
```

Figure 3.2: `contracts/Account.sol#L25-L28`.

This finding is considered informational because, at present, it appears that `AccountCreator` calls `init` immediately after an `AccountProxy` is deployed (see Figure 3.3 and 3.4). In such a case, there is no risk, because internal transactions are not susceptible to race conditions.

```
    function createAccount(address[] calldata _keys, address[] calldata _backups) external
onlyMultiOwners {
        AccountProxy accountProxy = new AccountProxy(accountImpl);
        initializeAccount(address(accountProxy), _keys, _backups);
    }
```

Figure 3.3: `contracts/AccountCreator.sol#L44-L47`.

```
    function initializeAccount(address payable _accountProxy, address[] memory _keys,
address[] memory _backups) internal {
        Account(_accountProxy).init(logicManager, accountStorage,
LogicManager(logicManager).getAuthorizedLogics(), _keys, _backups);
        emit AccountCreated(_accountProxy, _keys, _backups);
    }
```

Figure 3.4: `contracts/AccountCreator.sol#L32-L35`.

Exploit Scenario

Alice is a new MYKEY developer. Alice introduces code to MYKEY's backend servers that call `Account.init` directly. Eve notices this and starts racing those calls to `init`. Even wins several such races and uses her privileged access to those accounts to steal funds.

Recommendation

Short term, consider calling `init` from `Account` and `AccountProxy`'s constructors. This will prevent a race condition from allowing an attacker to set an account's `manager` field.

Long term, as new contracts are added to the codebase, ensure that they are properly initialized in their constructors. Doing so will avoid introducing race conditions into the codebase.

4. Use of `abi.encodePacked` could result in collisions

Severity: Informational
Type: Data Validation
Target: `contracts/AccountCreator.sol`

Difficulty: High
Finding ID: TOB-MYKEY-004

Description
The function `createCounterfactualAccount` uses `abi.encdePacked` with multiple dynamic arguments. Such uses could, in turn, cause calls to `create2` to try to create the same account and fail.

The beginning of the function `createCounterfactualAccount` appears in Figure 4.1.

```
    function createCounterfactualAccount(address[] calldata _keys, address[] calldata
_backups, bytes32 _salt) external onlyMultiOwners {
        bytes32 newSalt = keccak256(abi.encodePacked(_salt, _keys, _backups));
        bytes memory code = abi.encodePacked(type(AccountProxy).creationCode,
uint256(accountImpl));
        address payable accountProxy;
        // solium-disable-next-line security/no-inline-assembly
        assembly {
            // create2(endowment, mem_start, mem_length, salt)
            accountProxy := create2(0, add(code, 0x20), mload(code), newSalt)
```

Figure 4.1: `contracts/AccountCreator.sol#L57-L64`.

Note that arguments `_keys` and `_backups` are dynamic arrays passed adjacently to `abi.encodePacked`. So, for example, the following values of `_keys` and `_backups` would cause the call to `abi.encodePacked` to produce the same value:

- `_keys` = [X, Y], `_backups` = [Z]
- `_keys` = [X], `_backups` = [Y, Z]

The value that `abi.encodePacked` produces is passed to `create2`. If the same arguments are passed to `create2` twice, the second call will fail.

Note that a change to `createCounterfactualAccount` may also warrant a change to `getCounterfactualAccountAddress`.

The client points out that using a different salt in each call to `createCounterfactualAccount` mitigates this issue. We agree and have reduced the finding's severity to informational.

Exploit Scenario

Alice calls `createCounterfactualAccount` twice with values of `_keys` and `_backups` resembling the above bullets. Alice's second call fails.

Recommendation
Short term, change the uses of `abi.encodePacked` to `abi.encode` in `createCounterfactualAccount` and `getCounterfactualAccountAddress`. The changes to `createCounterfactualAccount` will eliminate potential collisions in calls to `create2`.

Long term, prefer `abi.encode` over `abi.encodePacked` whenever multiple dynamic arguments are involved. This will help prevent more of these collisions from entering the codebase.

5. Function `removeBackup` does not sufficiently validate backups

Severity: Medium
Type: Data Validation
Target: `contracts/logics/AccountLogic.sol`

Difficulty: High
Finding ID: TOB-MYKEY-005

Description
If a client wants to remove a backup from their account, they can do so using `removeBackup`. As shown in Figure 5.1, this function will set the expiration of the specified backup to `DELAY_CHANGE_BACKUP` seconds from the time that the function was called.

```
        function removeBackup(address payable _account, address _backup) external
allowSelfCallsOnly {
                uint256 index = findBackup(_account, _backup);
                require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");

                accountStorage.setBackupExpiryDate(_account, index, now +
DELAY_CHANGE_BACKUP);
                emit RemoveBackup(_account, _backup);
        }
```

Figure 5.1: `contracts/logics/AccountLogic.sol#L196-L202`.

However, there are three cases where usage of this function may result in unexpected behavior:

1. If a backup is currently pending removal, this function will delay the backup's removal.
2. If a backup has already been removed, this function will re-enable the backup.
3. If a backup has not yet been activated, this function will not remove the backup as expected. Instead, the backup will briefly be valid before expiring.

Exploit Scenario
Alice calls `removeBackup`, but then forgets that she did. After the backup expires, Alice calls `removeBackup` a second time, thereby reintroducing the backup.

Alternatively, Alice adds Eve as a backup. Halfway through the waiting period, Alice decides that she no longer wants Eve as a backup. She accidentally calls `removeBackup` instead of `cancelAddBackup`, which means that Eve will be a valid backup for a while after the waiting period has elapsed.

Recommendation
Short term, have `removeBackup` validate that the backup is eligible for removal. This consists of reverting when `_backup` is currently expiring or has already expired, and when the backup is not yet effective, as shown below:

```
uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_account, index);
uint256 expiryDate = accountStorage.getBackupExpiryDate(_account, index);
require(expiryDate == uint(-1), "expiring or already expired");
require(now >= effectiveDate, "backup not yet effective, use cancelAddBackup");
```

Long term, add tests for failing conditions. A more comprehensive set of unit tests could have exposed this bug (see also TOB-MYKEY-002).

6. Proposed method ID is calculated incorrectly

Severity: Low
Type: Data Validation
Target: `contracts/logics/DualsigsLogic.sol`

Difficulty: Low
Finding ID: TOB-MYKEY-006

Description
If a call to `changeAdminKeyWithoutDelay` is proposed using `DualsigsLogic` contract, the nonce is ignored when checking the signature. The function to extract the proposed function ID is shown in Figure 6.1.

```
    function getProposedMethodId(bytes memory _b) internal pure returns (bytes4 _a) {
                require(_b.length >= 164, "data length too short");
            // solium-disable-next-line security/no-inline-assembly
            assembly {
                        /* 'proposeByBoth' data example:
                        0x
                        7548cb94
// method id
                        000000000000000000000000b7055946345ad40f8cca3feb075dfadd9e2641b5
// param 0
                        00000000000000000000000011390e32ccdfb3f85e92b949c72fe482d77838f3
// param 1
                        0000000000000000000000000000000000000000000000000000000000000060
// data length including padding
                        0000000000000000000000000000000000000000000000000000000000000044
// true data length
                        441d2e50
// method id(proposed method: changeAdminKeyWithoutDelay)
                        000000000000000000000000b7055946345ad40f8cca3feb075dfadd9e2641b5
// param 0
                        00000000000000000000000013667a2711960c95fae074f90e0f739bc324d1ed
// param 1
                        0000000000000000000000000000000000000000000000000000000000000000
// padding
                        */
                // the first 32 bytes is the length of the bytes array _b
                        // 32 + 4 + 32 + 32 + 32 + 32 = 164
                _a := mload(add(_b, 164))
            }
        }
```

Figure 6.1: `contracts/logics/DualsigsLogic.sol#L158-178`.

This function assumes that the data array will start at the fifth word. However, that's not necessarily true. The third word stores the offset where the data array length is located, followed by the data array itself. If the offset is changed, the array position will change too.

Exploit Scenario
Alice constructs a malicious payload that masquerades as a call to `changeAdminKeyWithoutDelay`, but in reality calls a different function.

Recommendation
Short term, replace the existing use of `mload` with `mload(add(add(_b, /*32+4+32*/ 68), mload(add(_b, /*32+4+32+32*/ 100))))`.

Long term, consider using `abi.decode()` when parsing message data.

7. Unsafe storage load

Severity: Informational
Type: Data Validation
Target: `contracts/AccountProxy.sol`

Difficulty: High
Finding ID: TOB-MYKEY-007

Description
The implementation that the proxy will delegate to is loaded from storage using the code shown in Figure 7.1.

```
            assembly {
                let target := sload(0)
```

Figure 7.1: `contracts/AccountProxy.sol#L21-22`.

However, this is technically unsafe—slot 0 could have other data tightly packed into it as long as the data width is less than or equal to 96 bits.

Exploit Scenario
A new field is inserted after the implementation field. The code will now load an incorrect address from storage.

Recommendation
Short term, load the address outside of assembly and ensure that storage variables are not reordered, only appended to.

Long term, keep in mind how Solidity lays out storage variables when writing contracts using `delegatecall`.

8. EOS contracts do not build with the latest developer tools

Severity: Informational
Type: Patching
Target: Various

Difficulty: Low
Finding ID: TOB-MYKEY-008

Description
The version of the Contract Development Toolkit required to build the EOS contracts is 1.5.0, which is not the latest version. Since silent bug fixes are common, the contracts should be updated to build with the latest version.

The EOS contract build instructions appear in Figure 8.1. They make use of a docker container for `eosio.cdt` 1.5.0, which is from January 2019. The most recent version at the time of this writing is 1.7.0, which is from May 2020.

```
docker pull eostudio/eosio.cdt:v1.5.0

/* eostudio/eosio.cdt:v1.5.0 docker image id: 6248b2db433d */
docker run -v ~/mykey-eos-contracts:/tmp/mykey-eos-contracts -i -t 6248b2db433d /bin/bash

cd tmp/mykey-eos-contracts

./script/build-mgr.sh

./script/build-logic.sh
```

Figure 8.1: EOS contract build instructions.

Exploit Scenario
Eve learns of a vulnerability in `eosio.cdt` 1.5.0 and exploits this vulnerability in the EOS contracts, knowing that they rely on the old version.

Recommendation
Short term, update the code so that it builds with `eosio.cdt` 1.7.0. Using out-of-date dependencies could mean critical bug fixes are missed.

Long term, regularly monitor the `eosio.cdt` repository for updates. If an update is for a vulnerability, consider alerting your users and redeploying your contracts so your users do not rely on vulnerable code.

9. `LogicInitialized` events cannot be trusted

Severity: Informational
Type: Auditing and Logging
Target: Various

Difficulty: Low
Finding ID: TOB-MYKEY-009

Description
Each logic implementation must define a function called `initAccount`, which is called by newly created accounts. These functions typically all emit an event to indicate that the logic has been initialized for a specific account, as shown in Figure 9.1 for the `TransferLogic` contract.

```
    function initAccount(Account _account) external allowAccountCallsOnly(_account){
         emit TransferLogicInitialised(address(_account));
     }
```

Figure 9.1: `contracts/logics/TransferLogic.sol#L33-35`.

This function can be called by any address as long as they provide the same address as the `_account` parameter. This means the event cannot be relied on to provide correct information.

Exploit Scenario
Eve exploits a bug in an off-chain application by emitting unexpected initialization events.

Recommendation
Clearly indicate the potential to receive incorrect information when relying on the initialization events for the various logic contracts. Also consider indexing the `wallet` parameter in `BaseLogic`'s `LogicInitialized` event.

10. `DecodeBase58` implementation is not up to date

Severity: Informational
Type: Patching
Target: `accountmgr/accountmgr.cpp`

Difficulty: Low
Finding ID: TOB-MYKEY-010

Description
A function to decode a `base58` string was copied from the Bitcoin project. However, there have since been some modifications to Bitcoin's implementation, notably:

- A maximum return length parameter was introduced.
- A redundant loop was removed.
- Null bytes are properly filtered.

Exploit Scenario
A bug in the frontend causes the public keys to be corrupted with null bytes. The account manager accepts the corrupted public key instead of rejecting it.

Recommendation
Short term, update the `base58` decoder to the latest version.

Long term, monitor changes to the `base58` decoder so new patches can be merged.

11. Incorrect definition for `approveprop_action`

Severity: Informational
Type: Undefined Behavior
Target: `include/accountlogic.hpp`

Difficulty: Low
Finding ID: TOB-MYKEY-011

Description
The prototype for the function `approveprop` is given in Figure 11.1.

```
ACTION approveprop(eosio::name client, eosio::name backup, eosio::name proposer,
eosio::name act_to_approve, uint64_t index, std::vector<eosio::public_key>& new_keys);
```

Figure 11.1: `include/accountlogic.hpp#L53`.

The corresponding struct for this function, which is used to unpack input data, is given in Figure 11.2.

```
    struct approveprop_action {
        eosio::name client;
        eosio::name backup;
        eosio::name act_to_approve;
        uint64_t index;

        EOSLIB_SERIALIZE( approveprop_action, (client)(backup)(act_to_approve)(index) )
     };
```

Figure 11.2: `include/accountlogic.hpp#L143-L150`.

Note that the `proposer` parameter is missing from the struct.

Exploit Scenario
Code that attempts to unpack data for a call to `approveprop` results in undefined behavior, as the structure does not match the actual function parameters.

Recommendation
Short term, update the structure to include the `proposer` parameter.

Long term, write tests to ensure that all actions can be properly unpacked.

12. Dual-signed proposals cannot be canceled

Severity: Informational
Type: Denial of Service
Target: `contracts/logics/AccountLogic.sol`

Difficulty: Low
Finding ID: TOB-MYKEY-012

Description
Proposals can be created in two ways:

- A valid backup can propose a change to the admin key.
- The client and a backup can propose to change the admin key, change operation keys, or unfreeze operation keys without any delay.

If a backup creates a proposal, it can be canceled by the client using `cancelProposal`. However, as shown in Figure 12.1, `cancelProposal` does not support canceling a dual-signed proposal and there is no other way of canceling proposals.

```
        function cancelProposal(address payable _client, address _proposer, bytes4
_proposedActionId) external allowSelfCallsOnly {
                require(_client != _proposer, "cannot cancel dual signed proposal");
                accountStorage.clearProposalData(_client, _proposer, _proposedActionId);
                emit CancelProposal(_client, _proposer, _proposedActionId);
        }
```

Figure 12.1: `contracts/logics/AccountLogic.sol#L298-L302`.

Exploit Scenario
Alice is tricked into signing a proposal with a backup. She is unable to cancel the proposal, and her backups must be careful not to accidentally approve it.

Recommendation
Short term, provide a way for account owners to cancel dual-signed proposals, or ensure that all dual-signed proposals are cleared when the admin key is changed.

Long term, consider whether all workflows are implemented properly. Diagrams may help to visualize state transitions.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal

	implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Non–Security-Related Findings

This appendix contains findings that do not have immediate or obvious security implications.

- **The test "remove backup" does not property handle the case where the backup does not exist.** The test currently checks:

```
        await sleep(3000);
        var expiry = await accountStorage.getBackupExpiryDate(client, index);
        assert.equal(Math.floor(Date.now()/1000) > expiry, true, "remove backup
failed.");
```

  If there is no backup at `index`, then `expiry` will be 0, and the `assert` will succeed. Consider adding an additional `assert` before the call to sleep to verify that `expiry` is in the future.

- **Consider adding more helpful error messages.** In many places in the code, `require` statements like these appear:

```
require(success, "call to target failed");
require(success, "calling self failed");
require(success, "calling invoke failed");
```

  Such error messages do not communicate why the call failed. Consider appending the error message from the call's `returndata`. Note that this may require assembly.

- **Approval isn't registered when a proposal is overwritten.** When a proposal is overwritten, the approval array is reset but the approved backup that triggered the reset isn't added to the new array:

```
p.hash = _hash;
p.approval.length = 0;
```

- **`AccountStorage` could return tuples or structs.** Currently, a user who wants to get all the data for a backup key or a proposal must call one function for each property they want to read. In the future, consider providing a function that returns all fields at once.

- **Consider using error codes in return values.** Currently, magic values are used to indicate an error condition, such as a duplicate backup. In the future, consider returning a separate error value so the result of the function call is unambiguous to callers.

- **The error message in `DualsigsLogic.enter` contains the wrong function name.**

```
require(success, "enterWithDualSigs failed");
```

  There is no function named "`enterWithDualSigs`."

- **The `accountlogic.hpp` and `accountmgr.hpp` header files contain code.** These header files contain the dispatch functions for the `accountlogic` and `accountmgr` contracts, respectively. Since these functions are not required outside of the contracts' `.cpp` files, consider moving them there.

- **The order in which headers are included matters unnecessarily.** If `accountmgr.hpp` is included before `accountconfig.hpp`, errors such as the following result:

```
accountmgr/../include/accountmgr.hpp:81:32: error: use of undeclared identifier
'contractlst'
  typedef eosio::multi_index<N(contractlst), contract_lst> contractlst_index;
                                  ^
accountmgr/../include/accountmgr.hpp:87:32: error: use of undeclared identifier
'namelist'
  typedef eosio::multi_index<N(namelist), namerecord> namelist;
                                  ^
```

  Consider rewriting the headers in order to eliminate this dependency.

- **64-bit ints are cast to 32-bit ints in calls to `accountlogic::handle_deferred_tx_keydata`.** The function's `delay` parameter is a 32-bit integer:

```
void accountlogic::handle_deferred_tx_keydata(name to, name act, uint64_t
data_index, account_key key_data, uint32_t delay)
```

  However, in several places, the function is called with a 64-bit integer, resulting in a compiler warning. The actual integers involved do not seem to require 64 bits. Consider changing them to 32-bit integers.

- **The function `accountlogic::updatenonce` appears to be unused.**

```
void accountlogic::updatenonce(name to)
{
```

```
        require_auth(_self);
        print("updatenonce called.\n");
    }
```

Consider removing it.

D. Threat Model

This appendix describes our threat model for the EOS and ETH contracts. These models influenced the types of flaws we looked for in those contracts.

The EOS threat model is the simpler of the two. There are two contracts, `accountlogic` and `accountmgr`. They are both trusted, and interact with each other. An attacker, Eve, can interact with either of them (see Figure D.1).

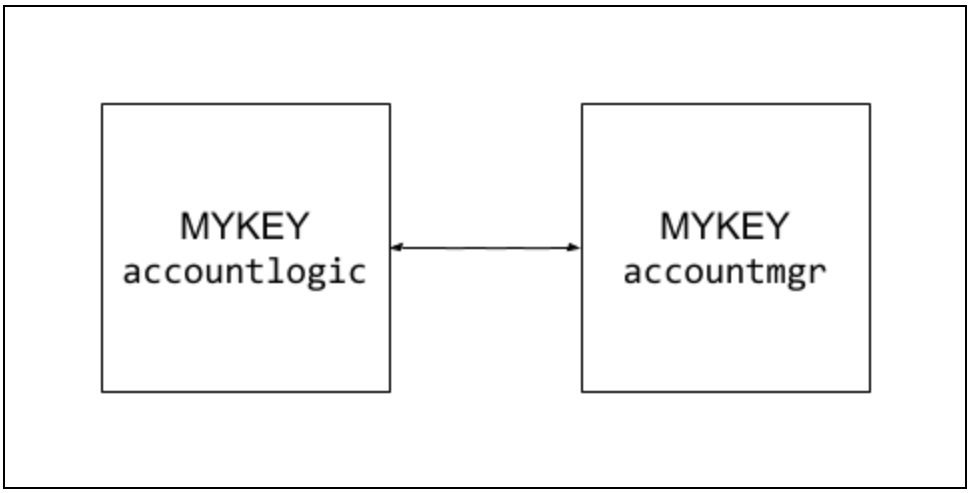

Figure D.1: EOS threat model.

The ETH threat model is more nuanced. Legitimate users, e.g,, Alice, have `Accounts` deployed through MYKEY's `AccountCreator`. As such, they reference MYKEY's `LogicManager` and trust the logic contracts that it manages. Alice's `Account` interacts with MYKEY's `AccountStorage` through those logic contracts.

Other users, e.g., Eve, may have `Accounts` that were not deployed through MYKEY's `AccountCreator`. In fact, Eve's "`Account`" may not even be an instance of MYKEY's `Account` contract. Eve's `Account` may reference her own "`LogicManager`," which manages her own logic contracts. Eve's `Account` interacts with MYKEY's `AccountStorage` through MYKEY's logic contracts or through her own (see Figure D.2).

With this model in mind, we considered it insignificant if Eve could adversely affect her own account data. For example, if Eve could corrupt her own keys within `AccountStorage`, we did not consider this an issue worth addressing. The only attacks that we considered meaningful were those where an attacker could influence another user's account data.

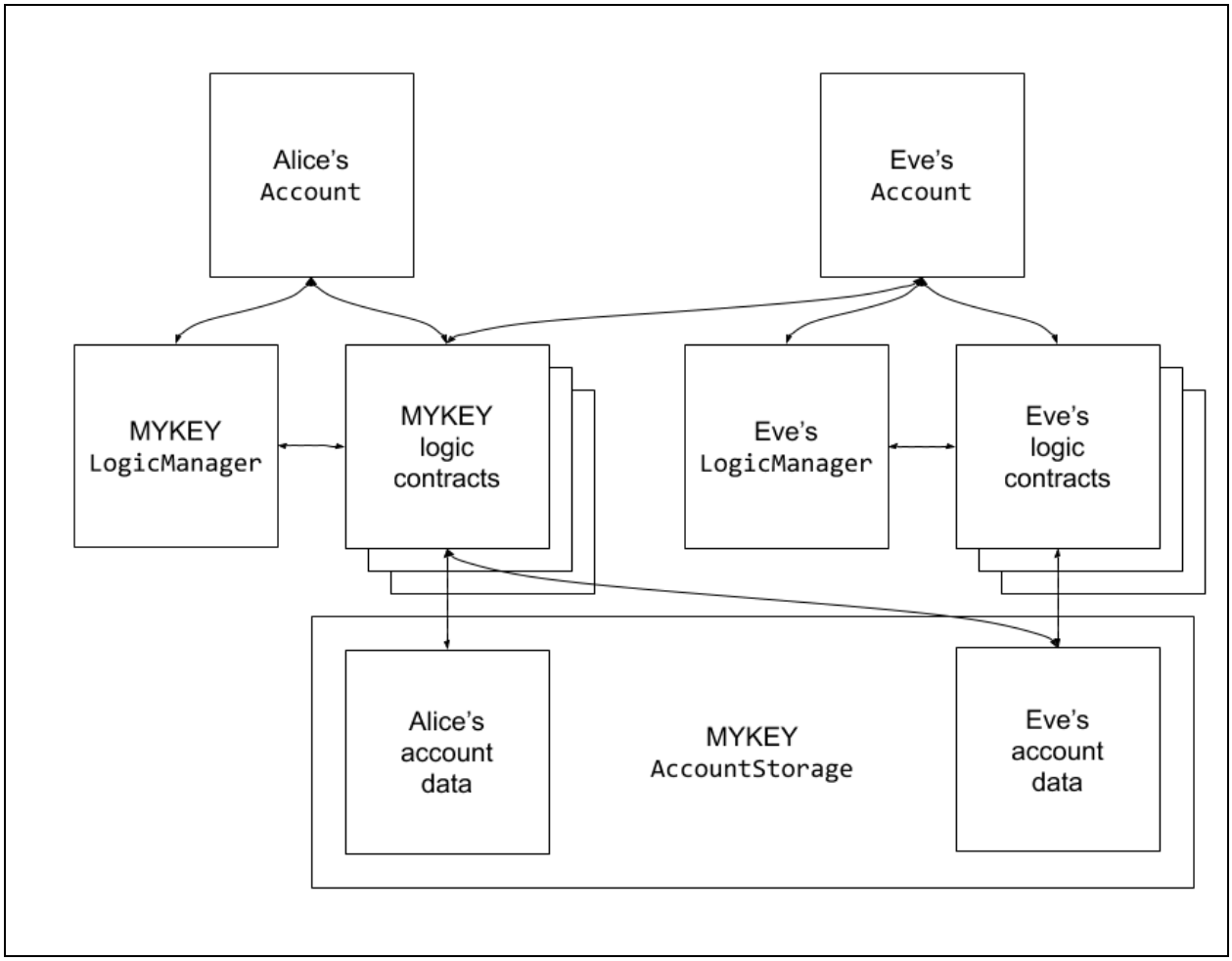

Figure D.2: ETH threat model.

A quick scan of `AccountStorage.sol` reveals that an account's data can be modified only by logic contracts that the account trusts (see Figure D.3, for example). This renders all attacks involving custom logic contracts moot. In other words, if Eve were to deploy her own logic contracts as in Figure D.2, it would not improve her ability to affect Alice's account data, since Alice's account would not trust Eve's logic contracts. Rather, the only way for Eve to influence Alice's account data is through the legitimate MYKEY logic contracts. Additional contract protection could help prevent such unintended modification.

```
    modifier allowAuthorizedLogicContractsCallsOnly(address payable _account) {
        require(LogicManager(Account(_account).manager()).isAuthorized(msg.sender), "not
an authorized logic");
        _;
    }
    ...
    function setKeyData(address payable _account, uint256 _index, address _key) external
allowAuthorizedLogicContractsCallsOnly(_account) {
        require(_key != address(0), "invalid _key value");
```

```
        KeyItem storage item = keyData[_account][_index];
        item.pubKey = _key;
    }
```

Figure D.3: `contracts/AccountStorage.sol#L14-L80`.

Users interact with the logic contracts through functions that make `this` calls. Generally speaking, the function making the `this` call performs data validation, serving as a kind of "gatekeeper." Modifications to user account data occur in the function called via `this`. This mechanism could be hardened by explicitly whitelisting the functions to which `this` calls are allowed. Note that the `ProposalLogic` contract already employs such a mechanism.

E. Fix Log

On August 18, 2020, Trail of Bits reviewed MYKEY's fixes for the issues identified in this report. The fixes to the EOS contracts were provided via Slack in a zip archive (`MYKEY-EOS-Contracts-ToB-review-20200731.zip`). The fixes to the ETH contracts appear in commit `8e7940f` on the `mykeylab/keyid-eth-contracts` GitHub repository.

MYKEY fixed or partially fixed seven of the 12 findings identified in this report. We reviewed the fixes to ensure they would be effective. Of the remaining five findings, MYKEY chose to accept the risk associated with two of them; two have not yet been fixed; and one was informational in nature and did not necessarily require a fix.

We noted that in some cases where risk was accepted or a fix was not yet applied, comments were nonetheless added to the source code to alert users to the risk.

After the fixes were implemented, MYKEY found a bug in the `TransferLogic` contract that made it behave incorrectly with respect to certain ERC20 tokens like USDT. MYKEY fixed the bug and asked Trail of Bits to review the fix, which appeared in commit `ec68db2`.

We tested the code in `ec68db2` with a USDT-like token whose `transfer` function never returned data. We also tested the fixed code with a BAT-like token whose `transfer` function always returned data, but never reverted. In each case, we observed the expected results, which suggests the fix was effective.

ID	Title	Severity	Status
01	Solidity compiler optimizations can be dangerous	Informational	Risk Accepted
02	Insufficient unit tests	Informational	Partially Fixed
03	Race condition on `Account.init`	Informational	Fix Not Required
04	Use of `abi.encodePacked` could result in collisions	Informational	Fixed
05	Function `removeBackup` does not sufficiently validate backups	Medium	Fixed
06	Proposed method ID is calculated incorrectly	Low	Fixed
07	Unsafe storage load	Informational	Risk Accepted

08	EOS contracts do not build with the latest developer tools	Informational	Not Fixed
09	`LogicInitialized` events cannot be trusted	Informational	Partially Fixed
10	`DecodeBase58` implementation is not up to date	Informational	Not Fixed
11	Incorrect definition for `approveprop_action`	Informational	Fixed
12	Dual-signed proposals cannot be canceled	Informational	Fixed

Detailed Fix Log

Finding 1: Solidity compiler optimizations can be dangerous
Risk accepted.

Finding 2: Insufficient unit tests
Partially fixed. Previously, none of the 98 `require` statements covered by unit tests were tested to fail. Tests have been added so that now only about two-thirds (66) of those `require` statements are not tested to fail.

Finding 3: Race condition on `Account.init`
This finding was informational in nature and did not necessarily require a fix.

Finding 4: Use of `abi.encodePacked` could result in collisions
The use of `abi.encodePacked` was replaced with `ebi.encode` in `createCounterfactualAccount`. The definition of `getCounterfactualAccountAddress` was adjusted accordingly.

Finding 5: Function `removeBackup` does not sufficiently validate backups
Fixed. The code in Figure E.1 was added to `removeBackup`.

```
        uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_account, index);
        // should be an effective backup
        require(effectiveDate <= now, "not effective yet");

        uint256 expiryDate = accountStorage.getBackupExpiryDate(_account, index);
        /*
         already expired: now > expiryDate;
         to be expired: now < expiryDate && expiryDate < uint256(-1)
         */
        require(expiryDate == uint256(-1), "already expired or to be expired");
```

Figure E.1: contracts/logics/AccountLogic.sol#L193-L202.

Finding 6: Proposed method ID is calculated incorrectly
Fixed. The proposed method ID is now computed as per our recommendation.

Finding 7: Unsafe storage load
Risk accepted.

Finding 8: EOS contracts do not build with the latest developer tools
Not fixed.

Finding 9: `LogicInitialized` events cannot be trusted

Partially fixed. MYKEY removed all “logic initialized” events except the one in `CommonStaticLogic`.

Finding 10: `DecodeBase58` implementation is not up-to-date
Not fixed.

Finding 11: Incorrect definition for `approveprop_action`
Fixed. MYKEY added the `proposer` and `new_keys` members to the struct.

Finding 12: Dual-signed proposals cannot be canceled
Fixed. MYKEY chose to clear all dual-signed proposals when an admin key changes.

Detailed Issue Discussion

Responses from MYKEY for each issue are included as quotes below.

Finding 1: Solidity compiler optimizations can be dangerous

> *We compared the compiling results with optimization on and off. During our test, it saves around 40% gas costs with optimization turned on. So we consider keeping it on.*
>
> *We will consider using* `solc`*.*

Finding 2: Insufficient unit tests

> *We added some unit tests for "unhappy" cases in ETH contract code.*
>
> *We will consider adding local test environment for EOS contract code.*

Finding 3: Race condition on `Account.init`

> *We make sure we use* `AccountCreator` *to create accounts.*

Finding 4: Use of `abi.encodePacked` could result in collisions

> *We have changed the uses of* `abi.encodePacked` *to* `abi.encode` *in* `createCounterfactualAccount` *and* `getCounterfactualAccountAddress`*.*

Finding 5: Function `removeBackup` does not sufficiently validate backups

> *We added* `require(expiryDate == uint256(-1), "already expired or to be expired")` *check to prevent reintroducing backup that has already expired or is going to expire.*
>
> *And we added* `require(effectiveDate <= now, "not effective yet")` *check to make sure only effective backups can be removed.*

Finding 6: Proposed method ID is calculated incorrectly

> *We replaced the expression with* `_a := mload(add(add(_b, 68), mload(add(_b, 100))))`*. It worked.*

Finding 7: Unsafe storage load

We added a comment in `AccountProxy.sol`*.*

Since the deployed `AccountProxy` *contract will not change, there won't be risk of incorrect storage load.*

Finding 8: EOS contracts do not build with the latest developer tools

We will consider upgrading to `eosio.cdt 1.7.0`*.*

Finding 9: `LogicInitialized` events cannot be trusted

We removed unused `AccountInit()` *functions and* `LogicInitialized` *events in several logics except that in* `CommonStaticLogic.sol`*.*

Finding 10: `DecodeBase58` implementation is not up-to-date

We will consider updating `DecodeBase58` *implementation.*

Finding 11: Incorrect definition for `approveprop_action`

We fixed this.

Finding 12: Dual-signed proposals cannot be canceled

In function `clearRelatedProposalAfterAdminKeyChanged()`*, we added* `clearProposalData` *to clear the other two types of proposals, and the* `isEffectiveBackup()` *check was removed.*